Escuela Politécnica Superior

Action Segmentation and Understanding in RGB Videos with Convolutional Neural Networks

Bachelor's Degree in Computer Engineering

Bachelor's Thesis

Author:
David Ivorra Piqueres

Supervisors:
José García Rodríguez
Alberto García García

Universitat d'Alacant
Universidad de Alicante

June 2018

UNIVERSITY OF ALICANTE

BACHELOR'S THESIS

Behaviour Segmentation and Analysis in RGB Videos with Neural Networks

Author
David IVORRA-PIQUERES

Advisors
Jose GARCIA-RODRIGUEZ
Alberto GARCIA-GARCIA

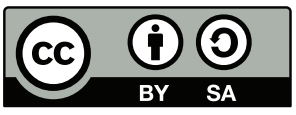
A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Bachelor's Degree in Computer Engineering
Department of Information Technologies and Computing

6th June 2018

This document was proudly made with LaTeXand Ti*k*Z.



'Science isn't about why, it's about why not'

Cave Johnson

Abstract

In this work, we propose three techniques for accelerating a modern action recognition pipeline. For reaching this point, we carried out an extensive study about the current state of action recognition. First, the traditional action recognition methods based on handcrafted features where reviewed, along with the ones focused on using machine learning as well as those which use deep learning techniques. Valuable insights were extracted from them, as well as the difficulties the task of action recognition carries out.

Subsequently, we explored numerous video datasets available for properly train deep models in the task of understanding human actions. Then, several video action recognition works from the past three years where thoroughly studied.

Coming back to the three proposed techniques, we first selected two of the reviewed deep learning works. Specifically, (1) Temporal Segment Networks (TSN), a Convolutional Neural Network (CNN) framework that makes use of a small number of video frames for obtaining robust predictions which have allowed to win the first place in the 2016 ActivityNet challenge; (2) MotionNet, a network that is capable of inferring optical flow from Red Green and Blue (RGB) frames by means of simple and transposed convolutions. As the third proposal we selected NVIDIA Video Loader (NVVL), a new Graphics Processing Unit (GPU) video decoding software developed by NVIDIA. Useful for efficiently reducing the data transfer and storage bottleneck of video deep learning applications.

Finally, we combine the first and third technologies. For this, we trained the RGB stream of the TSN network in videos loaded with NVVL on a subset of daily actions from the University of Central Florida 101 (UCF101) dataset. Thus proving their validity for their integration into a deep learning action recognition system that is both faster and effective.

Resumen

En este trabajo, proponemos tres técnicas para acelerar procesos de reconocimiento de acciones modernos. Para lograr esto, se ha llevado a cabo un estudio exentensivo del estado actual en el que se encuentra el campo de reconocimiento de acciones. Primero, los métodos tradicionales basados en la extracción manual de características han sido revisados, junto a aquellos que se centran en el uso de aprendizaje automático así como los basados en aprendizaje profundo. De ellos se han extraído conclusiones de gran interés así como las dificultades que la tarea del reconocimiento de acciones conlleva.

A continuación, hemos explorado un gran número de conjuntos de datos basados en vídeos para entenar correctamente los modelos profundos en la tarea de comprensión de acciones humanas. A continuación, multiples trabajos de reconocimiento de acciones en vídeos de los pasados tres últimos años, han sido estudiados en profundidad.

Volviendo a las tres propuestas, primero hemos seleccionado dos de los trabajos revisados de aprendizaje profundo. En particular: (1) TSN, un conjunto de herramientas y métodos para redes convolucionales que usa un pequeño número de fotogramas para obtener predicciones robustas, las cuales han servido para obtener el primer puesto en la competición de ActivityNet de 2016; (2) MotionNet, una red que es capaz de inferir el flujo óptico a partir de fotogramas RGB por medio de convoluciones simples y transpuestas. Como tercera propuesta seleccionamos NVVL, un nuevo software decodificar de vídeos por medio de GPUs desarrollado por NVIDIA. Útil para reducir eficientemente el cuello de botella que se suele formar en los procesos de almacenamiento y transferencia de datos a la hora de trabajar en aplicaciones de aprendizaje profundo orientadas a vídeos.

Finalemente, hemos combinado las tecnologías primera y tercera. Para ello, entrenamos la vertiente RGB de la red TSN en vídeos cargados mediante NVVL, pertenecientes a un subconjunto de acciones diarias del conjunto de datos UCF101. Demostrando de esta forma la validez para su integración en un sistema de reconocimiento de acciones basado en aprendizaje profundo más rápido y eficente.

Acknowledgements

In first place, I would like to thank you my Thesis advisors, Jose Garcia-Rodriguez for his motivation and guidance during all this year the developing of the Thesis has lasted, and giving me multiple opportunities for my future career. Also Albert Garcia-Garcia, for continually helping me from the beginning, whether it was from overseas or at the late hours of the night.

Finally, all of those who have borne me when I explained them what this Thesis was about, even the ones that had no idea of what I was talking about. You dared to ask!

Contents

List of Figures

List of Tables

List of Acronyms

2D two-dimensional

3D three-dimensional

ADL Activities of Daily Living

API Application Program Interface

Bi-LSTM Bidirectional Long short-term memory network

BN Batch Normalization

BP back-propagation

C3D 3-Dimensional Convolutional

CAM Class Activation Map

codec encoder-decoder

CNN Convolutional Neural Network

CNTK Computational Network Toolkit

CPU Central Processing Unit

CRF Conditional Random Field

CUDA Compute Unified Device Architecture

DNA Deoxyribonucleic Acid

DTW Dynamic Time Warping

FAIR Facebook Artificial Intelligence Research

FPS Frames Per Second

GPU Graphics Processing Unit

GRU Gated recurrent unit

HDD Hard Disk Drive

HMDB-51 Human Motion Database 51

HMM Hidden Markov Model

HTS Hidden Two-Stream

I3D Two-Stream Inflated 3D convolutional neural network

JHMDB Joint-annotated Human Motion Data Base

LAFF Local Accumulative Frame Feature

LSTM Long short-term memory network

MIT Massachusetts Institute of Technology

MMdnn Model model management, deep neural network

MNIST Modified National Institute of Standards and Technology

Mocap Motion Capture

MPII Max-Planck-Institut für Informatik

MS-LSTM Multi Stage LSTM

MSE Mean Squared Error

NVVL NVIDIA Video Loader

ONNX Open Neural Network Exchange

RAID Redundant Array of Independent Disks

ResNet Residual Neural Network

RGB Red Green and Blue

RGB-D Red Green and Blue, with Depth

RGB-diff Red Green and Blue Differences

RNN Recurrent Neural Network

SSD Solid State Drive

SSD Single Shot multi-box Detector

SSH Secure Shell

SSIM Structural Similarity

SVM Support Vector Machine

TCN Temporal Convolutional Networks

TSN Temporal Segment Networks

t-SNE t-Distributed Stochastic Neighbor Embedding

UCF101 University of Central Florida 101

UT-Interaction University of Texas Interaction

VGG16 Visual Geometry Group 16

YCbCr Y: Luminance; Cb: Chrominance-Blue; and Cr: Chrominance-Red

YOLO You Only Look Once

Chapter 1

Introduction

This first chapter introduces the main topic of this work. It is organized as follows. Section 1.1 sets up the framework for the activities performed during this thesis. Section 1.2 introduces the motivation of this work. Section 1.3 elaborates a state of the art of activity and action recognition systems their evolution since the past century. Section 1.4 lays down the proposal developed in this work and presents the main and specific goals of this project. In the end, Section 1.6 details the structure of this document.

1.1 Overview

In this Bachelor's Thesis, we have researched the task of action recognition from a computer vision perspective. The main approaches and datasets have been reviewed, at the same time that conclusions have been drawn from them. Moreover, we have centered our attention on the use of deep learning techniques for solving the task at hand, specifically with the use of CNNs. Finally, a series of methods and tool for accelerating a common action recognition pipeline have been proposed.

1.2 Motivation

This document presents the work that was carried out to prove the knowledge acquired during the *Bachelor's degree in Computing Engineering* taken at the *University of Alicante* between the years 2014 and 2018. The motivation of this work stems from diverse sources.

First, collaboration with the *Department of computer technology* is pursued. Concretely, helping with different machine learning and computer vision research tasks related with the national funded project, RETOGAR (RETOrno al hoGAR, in English, COMBAHO, COME BACK HOME) with reference code TIN2016-76515-R. Specifically, it is funded by *Ministerio de Economia y Competitividad* of the Spanish Government and counts with the participation of Jose Garcia-Rodriguez and Miguel Angel Cazorla-Quevedo as main researchers, both being professors at the University of Alicante.

From a personal point of view, deep learning, as a field of research and study, has always sparked my curiosity since I first heard of it. It demands for ingenuity, challenge, and scientific abilities in such a way that well suit the acquired background at the University and my own motivations. Indeed, the mathematical and logic foundations behind it, are what most attract me. In this Thesis I found an excellent opportunity for broaden in deep learning, and grasp some of its most advanced concepts.

1.3 Related Works

Although in recent years the task of activity recognition has witnessed numerous breakthroughs thanks to the development of new methodologies and the rebirth of deep learning techniques, the natural course of events has not always been like this. As for many years, despite of being tackled from multiple perspectives, the problem of constructing a system that is capable of identifying which activity is being performed in a given scene has been barely solved.

In this section, we will review the different approaches from which this problem has been faced. First, a brief introduction to relevant contributions of the last traditional methods (last years of the past century) and the first machine learning ones (first decade of current century) will be given. Then, the most recent advancements done in the field of deep learning that have been applied to the present task will be explained.

1.3.1 Handcrafted features dominance

It was near the beginning of the nineties when the interest for studying human movements and their interactions in a scene reached the computer vision community. This was motivated by the development of fundamental algorithms like optical flow extraction [1] or Canny edge detector [2] during the past decade (the 80s). Also, mathematical foundations carried out within the same time-frame, e.g., Hidden Markov Model (HMM) [3] or Dynamic Time Warping (DTW) [4] helped establishing a theoretical framework for the treatment of sequences, which consequently contributed to the processing of temporal series and modeling of a wide range of problems such as speech recognition [5] or Deoxyribonucleic Acid (DNA) treatment [6].

Forth in the last decade of the century, many of these methods had been applied to the problem of activity recognition. Conveniently, clear and in-depth examples can be found in "The Visual Analysis of Human Movement: A Survey" [7], where the author centers its attention on the review of hand and whole-body methods. Subsequently, four remarkable techniques are presented below.

Beginning with the pioneering paper "Recognizing human action in time-sequential images using Hidden Markov Models" [8], what can be considered as the first use of HMM for action recognition is exposed. Concretely, predictions are performed over image sequences. Low-level feature vectors of these images are extracted and transformed into symbols by means of vector quantization. Then, the classifier selects as the correct prediction from a group of HMMs (one for each action category) the one that best matches the input image sequence. An example of the model outputs can be found in Figure 1.1.

Although the use of low-level features contributes to the robustness of the model, they are easier to extract and sort out the reconstruction problems of geometric representation of human bodies (which are dispensable for solving the problem according to the author). It also has several drawbacks, such as in relation to the dimensionality where the process of describing action categories using low-level features is harder, as there is not an explicit relationship between the dimensions of the feature vector and the high-level descriptions of a category. Furthermore, the elevated number of dimensions in a sole vector makes the model representation process less intuitive.

Moreover, the carried out experiments report a drop on the accuracy of the model when trying to recognize actions performed by subjects different from the ones the HMMs have been trained with. This suggests either a lack of variability on the training

data (less than five subjects have been used) or that the extracted features, although effective, in this last aspect they are not powerful enough. The author points to the use of sensor fusion in order to improve the obtained results.

Symbol sequence	60 61 61 62 62 62 63 63 64 64 65 66 66 66 67 68 68 69 69 70 70 70 71 71

Figure 1.1: Low-level forehand volley sequence (top) with recognized symbols underlined (bottom). Figure extracted from [8].

In contrast to the use of two-dimensional features for representing motions, [9] proposed a three-dimensional (3D) structure of cylindrical connections for estimating limbs and joints of a human body and applied this representation to the modeling of walking pedestrians, which can be observed in Figure 1.2. Although similar works had been done, two contributions stand out. First, the use of ad-hoc data to help shaping the body motions (use of medical studies) allows to reach a realistic representation. Moreover, with this type of data, the author was able to sort out the problems of reconstructing human movements (the motion model) when using a kinematic approach, i.e. the system could keep a reduced number of parameters while realistic results similar to the ones of dynamic methods (which took into account forces and torques) could be provided. The other improvement is the ability to automatically determine the initial pose of the body. This is done by applying the 3D reconstruction procedure to every image, almost halving the (pose) search space after initialization.

Finally, the author identifies some limitations (and points out to its possible solutions) on the taken approach: (1) The need for the pedestrians to be parallel to the image plane, hence, the model is more sensible to variations on the viewing angle and occlusions, (2) experiments show that the overall quality decreases if real-world images are used in the model estimation instead of synthetic ones. One last issue is that only half of the human body shape is used for the model representation (as we present vertical line symmetry) in order to ease and speed up the model. The consequence of this approach is the inability to determine which element of any pair of limbs (arms, legs, feet...) is ahead of the other, increasing the confusion in the state of the walking cycle.

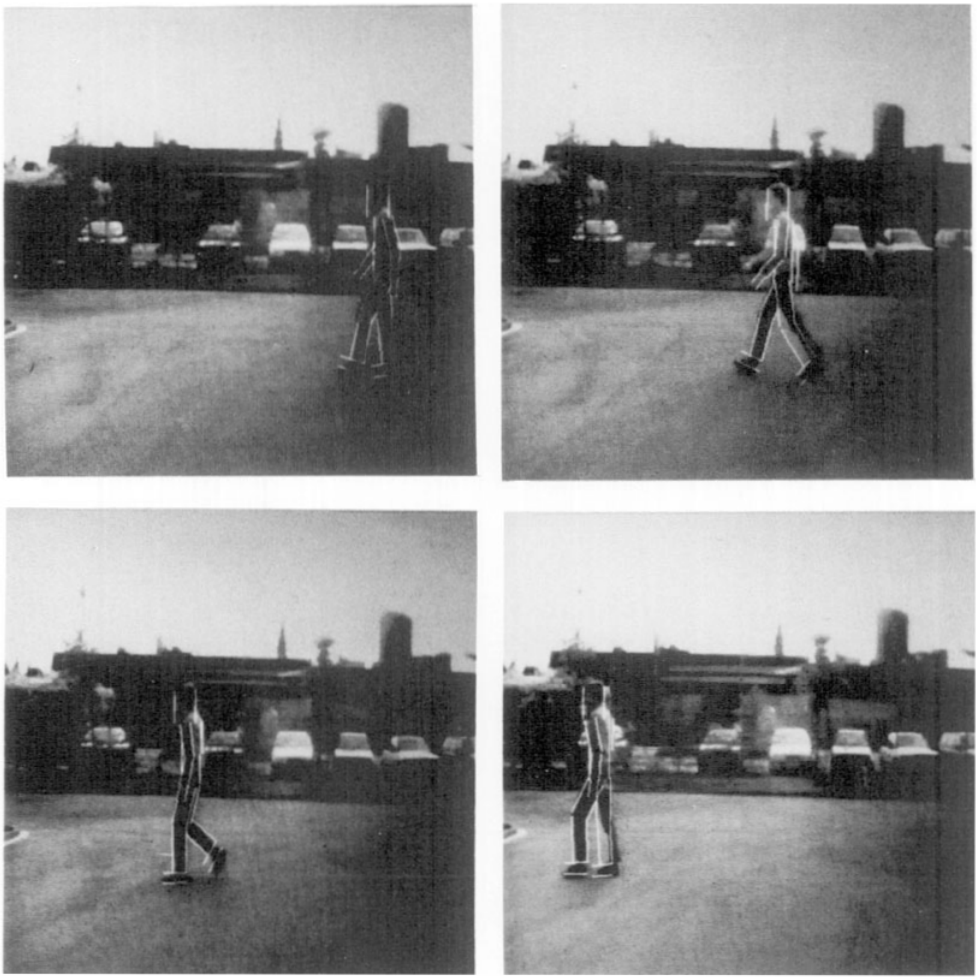


Figure 1.2: Estimated walking model, deviations between it and pedestrian can be noticed. Figure extracted from [9].

Another novel work of this decade can be found in [10] where one of the first uses of neural networks for activity recognition appears. Concretely, the authors prove that it is possible to use a back-propagation (BP) neural network for categorizing human motions, three classes are used for this purpose: walking, running, and other motions. In this occasion, each body is represented by a skeleton of bi-dimensional segments, obtained from the silhouette of the corresponding moving subject. Although authors could have used a Recurrent Neural Network (RNN) for the recognition task, they opted for converting the sequence of moving figures into a series of Fourier transforms in which the most meaningful components are extracted and combined with directly-obtained skeleton features for serving as input to the feed-forward network. Moreover, with only 22 features and a simple three-layer network model, which can be observed in Figure 1.3, authors were capable of obtaining great performance (over 90%). This is something remarkable for the period of time in which this system was developed. Nevertheless, the reported results can be questionable, as very few data points are used in the test phase. Furthermore, one more aspect worth mentioning is the labor made on the preprocessing step, as researchers greatly reduced the complexity of the data at the same time the most relevant information was kept.

On the other hand, the model selected for the representation of motion (figure segments) lacks the ability to fully capture sophisticated motion patterns. For this, authors aim to experiment with a 3D model in future research. Additionally, as in previous works, subjects still need to be parallel to the image plane for correctly classifying their motion. Finally, authors state that they were able to find the correct number of nodes in the hidden layer of the network just by sole experimentation. Specifically, only with two nodes, the network diverged, and with three, convergence was still a complicated issue. At the end, it was with one more node, with which convergence was possible, though, researchers were unable to explain why this configuration was a successful one.

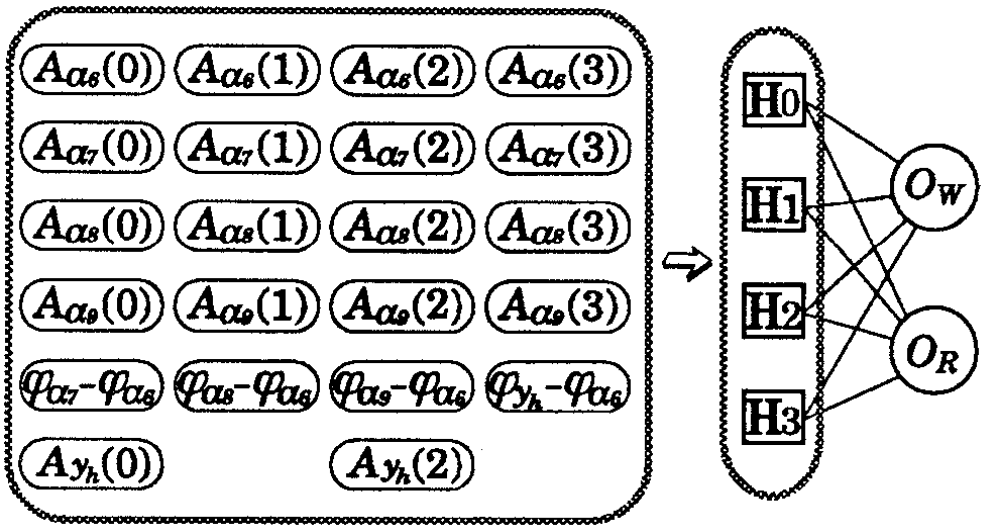


Figure 1.3: Three-layer network utilized. From left to right: input layer, hidden layer and output layer. With 22, 4 and 2 parameters respectively. Extracted from [10].

Last relevant technique reviewed for this decade is [11], where a fully mathematical approach is taken for the task of recognizing human motion patterns. More in detail, the presented model is applied to the classification of ballet steps, since they are variated and many examples exist. For the representation, three-dimensional data points (coming from the tracking of different dancers) are processed for the creation of a phase-space which represents the different states in which the dance movements can be. After adjusting the model parameters via a supervised method and creating the dance step curves in the phase space (observed in Figure 1.4), predictions are made in terms of proximity between the input data and the computed curves. A particularity of this approach is that since the used features are simply Cartesian coordinates, the system is not directly dependent on time data (phase-space is constructed in terms of velocity). This complexity is reduced at the cost of an increase in the probability of mistakenly assigning a class to a step that does not belong to it. Nevertheless, this does not seem a determining factor for the final system accuracy.

Furthermore, as a consequence of the selected approach, different classes for one ballet movement can be recognized at the same time, as constrains in the model have not been imposed to prevent this. Other existing issue is that, due to limitations on the size of the test set, researchers had to reuse part of the training samples during

the evaluation phase, making the reported results less accurate. On the other hand, although the system was largely trained with atomic ballet steps, authors report success in correctly classifying more complex movements (derived from the combination of the previous ones). Finally, researchers intend to improve the system by making it differentiate between the basic parts that compose a complex movement.

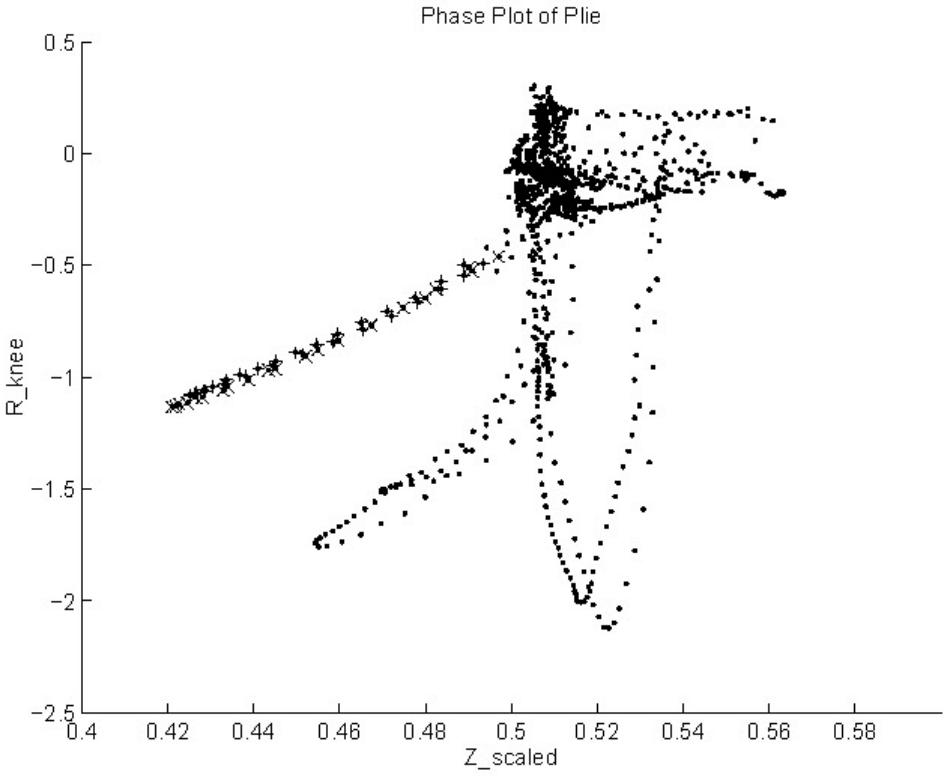


Figure 1.4: Movements in 2D phase space. Symbols "x" and "+" mark time steps of Plié movement for two dancers, "." marks other movements. Plié lays in a distinctive region from other movements. Figure extracted from [11].

1.3.2 New machine learning approaches

In contrast with the past ten years, where the topic of activity recognition was in its beginnings, few approaches existed and those relevant ones provided nearly toy results, in this period of time, activity recognition started to flourish. Not only this problem was attacked from a wide range of different fronts, but also a more general trend on the use of well-known machine learning techniques as the basis of its solutions was followed. Clearly, many are the relevant works of this period and in the following lines those significant to the problem at hand are analyzed.

One of the known challenges of this period was the difficulty to obtain proper features from images of moving subjects that were too far, partially occluded, or viewed from difficult angles, such as when motion is not parallel to the image plane. For this, [12] make one of the first steps towards finding a solution to this problem by identifying distant motions and obtaining spatial patterns through optical flow extraction.

First a subject is tracked through the image sequence of a motion, creating a spatio-temporal volume which is stabilized and corrected in order to have the human subject at the center of each image element of the volume. After this, optical flow is computed for every frame of the sequence (volume), with the advantage of extracting knowledge from the relative motion of the different human limbs (foots, arms, head...), being captured over the course of the sequence. Finally, the extracted optical flow is used like a template for being matched during the selection process of the motion category. For this, it is required to be resistant against data that presents noise or that it is significantly unaligned, either in the spatial or the temporal axis. To accomplish it, blurring is proposed for obtaining the final features, with a previous step where the volume is divided in different information channels in order to avoid the lose of any essential information after the blur modification process. For classifying a motion, the aforementioned features, named "spatio-temporal motion descriptors" are compared with stored sequences of each class. This is done in terms of frame similarity, computing a correlation function for the whole extent of the sequence. The final decision is obtained from the highest ranked label of a k-nearest-neighbors algorithm.

The system is tested on three different action datasets: ballet, tennis, and football. Although at first sight results do not seem very relevant, the overall low quality of the sequence images should be appreciated, making the results more valuable. It is also interesting to see that the taken approach is able to interpret the direction of motion, e.g., running to the left or running to the right, despite the lack of movement information of the subject w.r.t the background (subject-centered frames). Thus, as previously mentioned, the method takes the needed knowledge from the various body parts of a tracked individual.

Looking at the applications of the developed system, researchers propose interpreting the recognition task as a query made to a database, where the demand content is the input sequence and the objective is to find the most representative label among a bank of class instances. With this idea in mind, they present three types of functionalities, which can be summarized in the Figure 1.5. The first one is the recovery of the annotated body skeleton of the best matching sequence along with the sequence itself. Furthermore, this skeleton can be transfered to the queried motion by assigning each instance of the skeleton to a frame of the input sequence. This can also be done with three-dimensional skeletons, whether by motion capture and subsequently two-dimensional (2D) rendering (extending the dataset with synthetic sequences) or by converting the bi-dimensional skeletons into 3D ones. This can lead to unclear samples since in some cases is difficult to determine in an image which element of a limb pair, like legs or arms, is the one moving. The second functionality is related to video synthesis, where the objective is to create a new frame sequence performed by a selected subject, either by a succession of commands (class labels) or by matching the motions of another subject. Both options base its algorithm in retrieving the information from the database similarly to the previous query-based operations in conjunction with dynamic programming. Last functionality is the possibility of removing artifacts, such as occlusions or cluttering, from a human motion sequence. This is accomplished by: (1) Finding for each input sequence frame the most similar frames on the database, (2) obtain the median frame from the similar retrieved ones, (3) use it as the new corrected frame. For obtaining proper results a large amount of representative and variated data is needed, since the principle is that with the median, the similarities between input and extracted frames (the parts that should be kept) and the variations (the not relevant ones) will be determined and properly corrected.

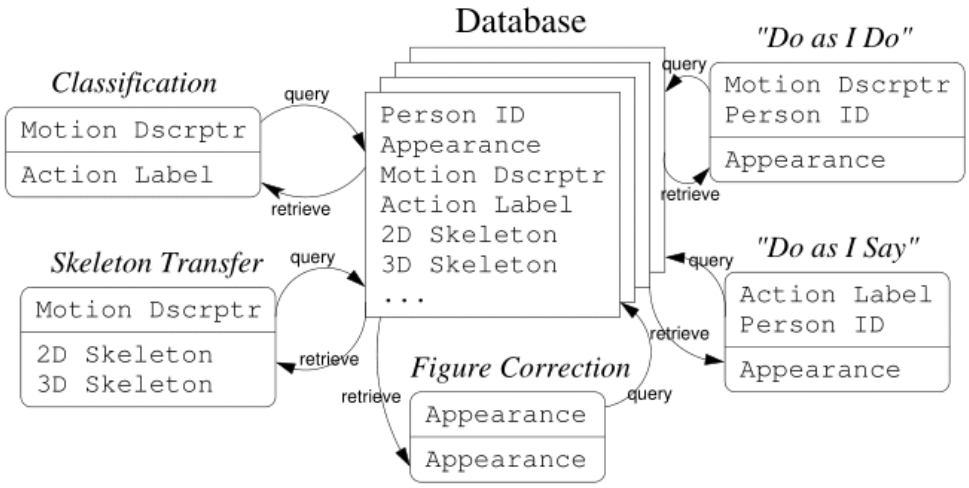


Figure 1.5: Scheme of the database approach together with its applications. Figure extracted from [12].

Also remarkable is the investigation carried out by [13], where apart from contributing to the computer vision community with the KTH dataset, authors recognize motions by taking an almost only machine learning approach. Specifically by extracting spatio-temporal features of a video and using them for training SVMs classifiers. The characteristics of the features, which can be observed in Figure 1.6, is their locality (local features), which enable them to: (1) Be robust against image perturbations in videos, (2) gain independence upon the image capturing process and its circumstances, since they can conform to characteristics of local motions by adjusting specific parameters without many difficulties. Results on a newly created dataset, confirm these. For example, when compared to other kind of features, like basic spatio-temporal histograms, local ones perform better at locating motions. Even with the ones that, while pertaining to the same class of action, are performed with multiple illumination conditions, different speeds and scales or even by distinct subjects that change in appearance between samples. In addition, the selection of the correct kernels for each type of feature plays an important role on the given SVM classification results, even more for local features, where a kernel specifically designed for its processing known as "local kernel", is applied.

Despite the use of locality improving performance, recognition errors that look similar, are found through the different tested approaches. For example, when looking at the false positives given by the model to certain action categories, we find that similar actions still tend to be difficult to distinguish. This is the case for pairs of classes like hand-waving and boxing or walking and jogging. Also, as global frame data is mostly discarded in the feature extraction process, when finding correspondences between related motions difficulties re-appear. Thus, leading to locate a specific feature, for example present in two sequences, at incorrect areas in some of its frames. Moreover, for increasing the quality of the model prediction, authors suggest the possibility of combining local features with other ones due to the existence of relationships between them, space and time.

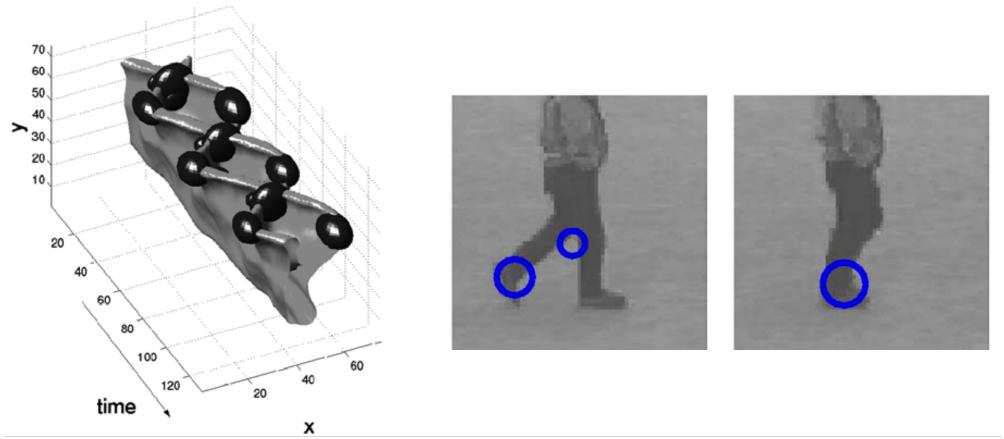


Figure 1.6: Spatio-temporal walking features visualization and its correspondence. Figure extracted from [13].

Differentiating from the researches at the time, where recognition models required to process sequences of several frames for being able to make accurate predictions, the system developed on the work of [14], only takes into account up to 10 frames of a recorded motion. The main objective of the investigation is to explore how much visual information is needed for a model to properly recognize target motions. Initially, form (through edge detection) and optical flow features are extracted for each frame of subject-centered sequences, and processed independently in a similar manner following a two stream approach. For the first type of features, down-sampling via max pooling is applied in order to reduce their dimensionality as well as to gain higher resistance to different subject in-frame translations. Moreover, the extracted forms are compared with PCA vectors, computed during the model training, in a form of template matching process, yielding as result a set of similarity values for the frames taken into consideration. The other type are optical flow features computed only from the previous frame of a given one. The flow follows the same process than the form features, both for down-sampling and for template matching. After obtaining the similarity vectors for the two types of features, they are weighted and concatenated into a large dimensional vector. For the recognition task this vector is used with a group of binary linear SVM classifiers, one for each possible action class. This process is represented in Figure 1.7.

At the time of selecting which of the two similarity vectors should receive more importance in the concatenation, experiments show that the decision is more dependent on the target task rather than the information the vectors carry. This was determined after looking at what happened if each vector was zero-weighted. For some datasets it was the former type which outperformed the later while in others the opposite happened. Indeed, it was the combination of both which gave the best results. Also, authors looked upon improving the system by making it more resilient to a number of invariance types such as rotation or scale.

On the question of how many frames where needed for the recognition of actions, authors found that with only seven frames, results where on par with the best ones of the time. Moreover, they noticed that if the number of frames was increased up to 10 frames, performance started to drop slightly.

After testing the method on full motion sequences, authors reported good quality results. Nonetheless as stated in the publication, there is not a standard way on how to

report results, serving this more for demonstrating the capacities of the system rather than as fact. Still for researchers, which is the more relevant set of frames in a sequence and which should be the smallest meaningful unit of an action, is an open question.

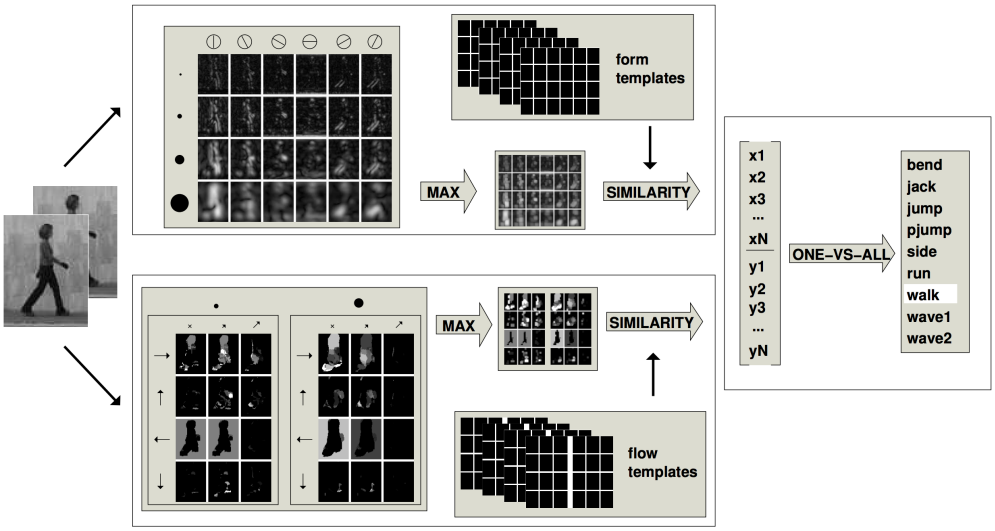


Figure 1.7: Recognition system pipeline. Form and flow features are extracted and max-pooled. Similarity vectors from both are obtained from template matching and concatenated. Finally, binary SVMs are used for classification. Figure extracted from [14].

While at the time of developing systems that fully recognize what is currently happening in a video is common to divide the problem in independent tasks like action classification or object recognition, [15] investigates them jointly. Concretely, the relationship between actions and scenes in videos, since in many real-world events the former tend to take place in the same types of the latter. For this, a model that mutually predicts both is presented.

First of all, action segments are automatically extracted from movies by making use of script and subtitles alignment (text mining). This procedure enables authors to selectively create their training and testing datasets. A similar method is used to retrieve scenes and find the ones that co-occur with the selected actions. Once the data is gathered, experiments are run over the two datasets independently. For both, subject movement and static features are represented by a bag-of-features. A series of binary SVMs classifiers are used for predicting action and scene labels separately. The choice of using this type of classification instead of a multi-class one is due to the fact that each one can be detected without depending from the others and different labels can appear simultaneously. For example, two distinct actions can be performed inside the same prediction time frame. On the results, data shows that although sometimes the use of static features can be beneficial, in general the contrary happens. On the other side (like in previous reports) the combination of both types of features improves performance for the two classification tasks.

After this, action SVMs are adjusted for using information obtained from the scene classifiers in order to gain advantage from the context where they takes place. A sample plot of the results can be observed in Figure 1.8.

It is clear that actions that occur in a variety of situations do not see their classification accuracy greatly increased, while the ones dependent on the context receive a

considerable improvement. Finally, researchers briefly present results with the opposite method, classifying scenes with the help of action information. Similarly, performance is enhanced by action context (help to determine what actions can happen in each scene) reinforcing the usefulness of knowledge combining for the whole task of video understanding.

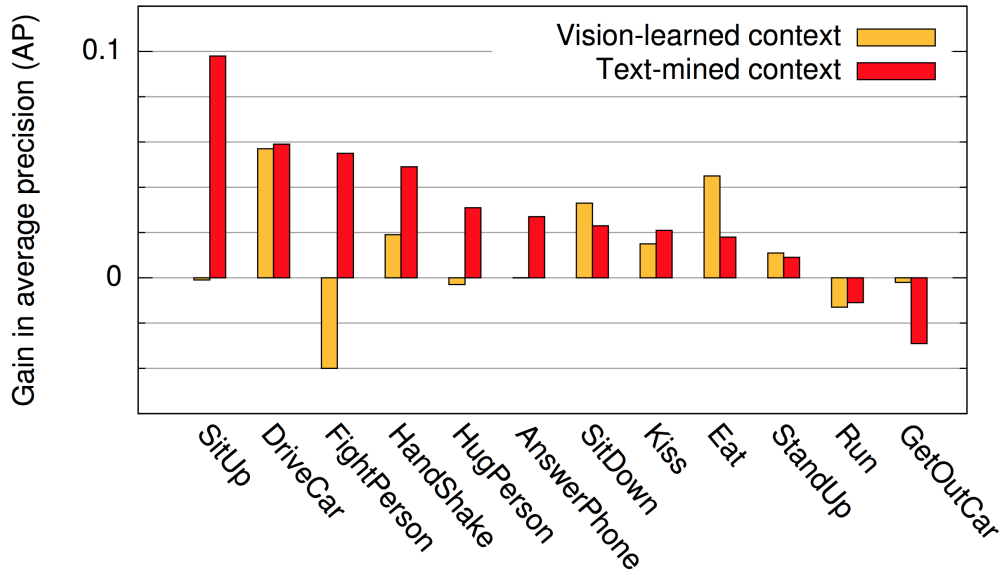


Figure 1.8: Action recognition improvement from scene (up to 10%) and text mining (around 2.5%) contexts. Actions that occur in a specific scene, like *driving a car*, get a substantial gain. Conversely, the ones that happen almost anywhere improve less, e.g *standing up*. Figure extracted from [15].

1.3.3 Second rise of deep learning

Following the same line of investigation started during the initial years of this century, in which machine learning has been extensively used, the deep learning branch of techniques has served during the most recent years and until today, as a great tool when confronting to the problem of action recognition by both its efficiency and its effectiveness. Consequently, attention is paid to remarkable achievements accomplished within this context.

Differently to many deep learning systems for action recognition in videos in which spatial features from still frames are extracted and passed to other type of models such as recurrent or dense networks, in the work of [16], three-dimensional convolutions are proposed for jointly compute spatial and temporal features from various subsequent frames. Moreover, each channel of the convolutions has an independent processing (pooling and successive convolutions), allowing to extract a wide range of motion features for each individual frame at the same time. Specifically, the proposed 3D convolutions make use of cubic kernels that are convolved over a defined number of consecutive frames (later, feature maps). As in two-dimensional convolutions, different groups of kernels can be applied to the same input feature maps in order to output a wide range of newer ones. This type of operation enables authors to experiment with

a number of model configurations. Concretely, they construct a three layer network which takes as input segments of seven frames into five feature channels (gradient and optical flow of the spatial axes together with gray pixel values). AN overview of the architecture can be observed in Figure 1.9.

In general, this allows them to obtain better performance than only using RGB information. Particularly, as the temporal length is small after the second convolution and pooling (two for optical flow and three for the rest of feature maps), only 2D convolutions are applied in the third convolutional layer. All the feature maps are transformed into one single feature vector, which is taken by a fully connected layer with the same number of neurons as its length, in order to assign the corresponding class to each action. For further improving the performance of the model and not only limit the knowledge of an action to a group of seven frames, a feature vector extracted from the whole sequence, representing high-level motion information, is used as a regularizer for the inputs of the model by bringing closer to it, during each iteration, the feature vector of the last hidden layer.

For the different experiments two action datasets are used, TRECVID 2008 and KTH. In each of them, one-versus-all classification is applied. Moreover, with the first dataset the effect of different 3D convolutional architectures and how they act when evaluated as one single model is investigated. The results report an increase on the performance when combining the outputs of the model variations. This holds true even if their individual performance is of a less richer quality. These, shows that complementing information from different models can lead to great improvements in the recognition task.

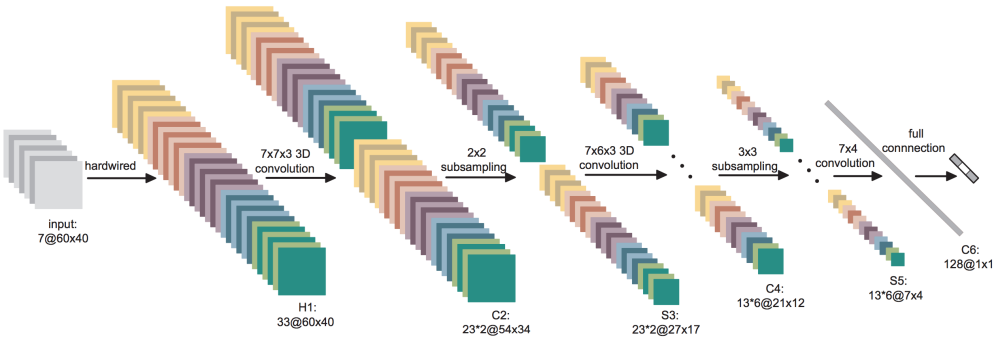


Figure 1.9: Three-dimensional convolutional architecture used for action recognition. Figure extracted from [16].

One of the most common approaches in recent deep learning architectures is the combination of independent information streams on the same model. For the case of action recognition, one of the most successful strategies is to process action sequences through two types of streams, spatial and temporal. The first uses single frames for the recognition of shapes and to obtain context information, while the second, serves for the recognition of motion through frame sequences. In the work of [17], one of the first systems that follows this approach is presented, an overview of it can be observed in Figure 1.10. Moreover, it makes use of two-dimensional convolutions and subsequent dense layers for the two streams. At the time of combining its scores, two options are investigated: (1) Serve the scores to a multi-class SVM, (2) directly average the scores. After analyzing the results, the second method presents a more satisfactory performance.

Firstly, the spatial stream acts similarly to an image classifier since working with single frames. As authors suggest, its efficiency can be improved by pre-training it with large-scale variated datasets like ImageNet, which is used in the reported experiments. On the other hand, for the temporal stream the input is conformed by several channels of optical flow based features, computed from two subsequent frames. The first variation, computes the horizontal and vertical optical flow components. The second, bases the feature information in the trajectory of the pixels. The third, a bidirectional version (by computing the flow in the forward and backward time directions) of either the first or the second variations. Lastly, the fourth is an optical flow centered at the origin (zero) by removing the motion of the camera. For better training the temporal CNNs, since they work with video sequences and need a good number of samples, the network makes use of the UCF101 and Human Motion Database 51 (HMDB-51) datasets at the same time. However, as both do not share the same class representations, for using the two datasets without restrictions the temporal CNNs is modified for having two softmax output layers, working each layer on one specific dataset. The addition of the two dataset specific losses is the temporal stream final loss.

At the time of training the networks, the spatial stream receives one random frame from each video sample at each iteration. In the case of the temporal stream, the input volume is composed of frames subsequent to the one randomly extracted for the spatial stream. Here, the different optical flow based features are tested resulting more beneficial the fourth variation (optical flow with the motion of the camera removed). Also, as the temporal stream receives motion information, it independently performs better that the spatial stream alone. In addition, clearly the combination of both streams gives better results than the two streams alone, since the overall representation is improved by the integration of the information obtained from each independent stream.

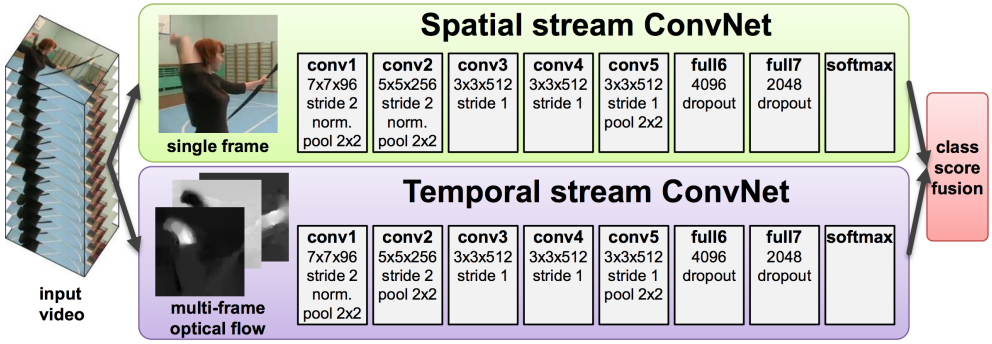


Figure 1.10: Representation of the two-stream model. Figure extracted from [17].

One interesting approach for obtaining signifying features of actions in video sequences is presented in the work of [18] where specific subject pose-features are extracted, for different parts of the body from each frame, through the use of CNNs by taking into account its motion through the subsequent parts of the sequence. The approach is summarized in Figure **??**. More in detail, optical flow (horizontal and vertical axes) and magnitude components are extracted for each two consecutive frames of the sequence, acting as a image with a volume of three components. Independently from this step, for every frame the skeleton of the subject and its parts are detected. Then, for both the RGB and optical flow components images, the sections of the hands, the upper part of the body and the whole body, are extracted as separate images. These and the whole

image are processed by two specialized CNNs: One for the RGB based images and the other for the optical flow based ones. For representing the temporal component of an action sample, for each sequence of images of each type (extracted and whole image) researchers obtain the feature vectors in each CNN, corresponding to the layer previous to the output one. The temporal components consist of two parts: (1) The static part, a vector with the concatenation of the minimum (absence of detection feature) and maximum (presence of detection feature) values of each dimension among the previously mentioned feature vectors (independently for the RGB images and the optical flow ones); (2) the dynamic part, a vector of the same type as the static part, but with the minimum and maximum for each dimension resulting from element-wise subtracting the components of a current feature vector from the one corresponding to four sequence elements ahead. This is done for each vector of the sequence. At the end of this process, the pose feature vector is composed by the concatenation of all the temporal component values (from both optical flow and RGB parts) of all the image types, having been these values previously normalized. Appearing first in this vector the temporal components of the RGB based images and latter the ones of the optical flow based images. Finally, this feature vector is processed by a linear SVM which carries out the recognition of each action sample.

Through different experiments, results show that the combination of all the body part images perform better than only using them independently. Also, the joint use of optical flow and RGB based images as well as the application of the absence of detection features improves the final quality of the model. Furthermore, the authors show the great resistance of the pose features to errors in the extracted poses. These errors are computed as the difference between the performance obtained from the model for the Joint-annotated Human Motion Data Base (JHMDB) dataset samples with the dataset-proposed poses and the performance obtained from these samples but with the poses extracted manually. Moreover, the same errors are measured in a subset of the Max-Planck-Institut für Informatik (MPII) Cooking dataset, where two fine-grained action classes (wash hands and wash an object) have been defined by hand. Finally, the overall results sign to the importance of taking into account the poses of the subjects at the time of confronting the task of action recognition and the ability to better improve the capacity of a system for identifying low-level action classes.

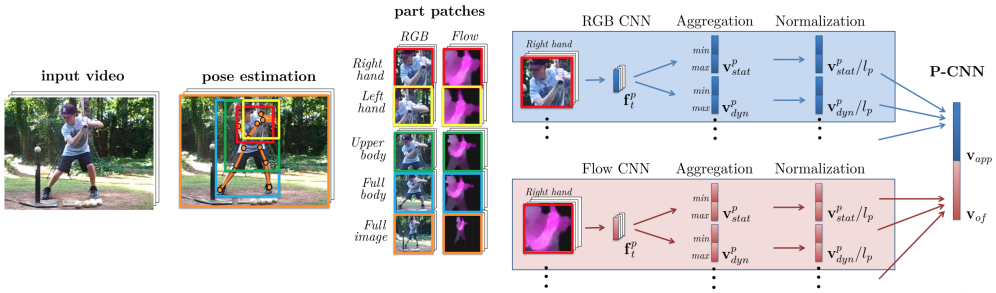


Figure 1.11: Pose-feature extraction process: First, pose is estimated and different body image patches are extracted. Flow and RGB features are extracted from those through CNNs. Finally, maximum and minimum detection is applied, features are aggregated in a vector, the resulting vector is the concatenation of the two types of feature vectors after normalization. Figure extracted from [18].

1.4 Proposal and Goals

The main purpose of this project is to present improvements for the developing of a system that allows to enhance the autonomy of people with acquired brain injury who are dependent in their integration in society. For accomplishing this goal, a model that is capable of recognizing the actions of a subject, operating in real-time with a robust accuracy, has to be developed.

For this reason, a study of the current deep learning methods applied to action recognition, and their related datasets with be carried out. Furthermore, the most recent technologies and systems oriented to real-time action recognition with deep learning, and video processing will be analyzed. Finally, the most promising ones will be tested and proposed for its incorporation in order to accelerate a common action recognition system.

1.5 Schedule

The proposed goals will be carried out during a timespan of about 9 months. Starting at the beginning of September, day 15th, with an 150 hours internship (IN) at the *Department of computer technology* and finishing on the 4th of June with the final document review and submission. In general, the workload will be distributed between three days each week. Two lecture days and one in the weekend. An overview of the tasks in which the project is divided, can be observed in Figure 1.12.

Figure 1.12: Timeline overview.

Subsequently, each of the four tasks is divided into specific ones. The first one, **1. Practical introduction**, lasts 48 days, from the 15th of September to March 21st. It consists of:

- **1.1 Review deep learning core concepts.**
- **1.2 Complete Keras and PyTorch tutorials.**
- **1.3 Learn about action recognition approaches.**
- **1.4 Write document Motivation and Proposal sections.**

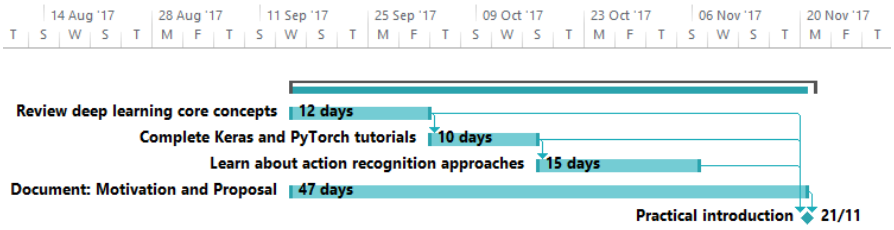


Figure 1.13: Task 1 Gantt diagram.

The next task, **2. State of the art review**, starts on November 22nd and ends on February 2nd, with a total of 53 days. It is divided in:

- **2.1 Sweep existing approaches in computer vision and deep learning conferences (ICCV, ECCV, NIPS, and BMVC).**
- **2.2 Look up dataset compilations and surveys.**
- **2.3 Test video classification methods[1].**
- **2.4 Write State of the art draft.**

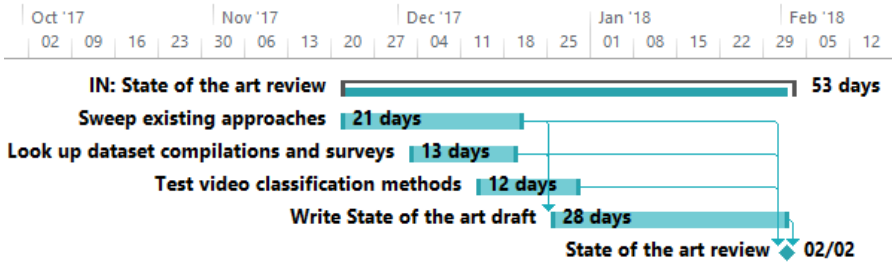


Figure 1.14: Task 2 Gantt diagram.

The third task, **3. CNN architecture study**, starts on February 5 and ends on March 28, with a total of 38 days. It is split in:

- **3.1 Run and check Multi Stage LSTM (MS-LSTM) model results.**
- **3.2 Run and check TSN model results.**
- **3.3 Run and check Hidden Two-Stream (HTS) model results.**
- **3.4 Write Methodologies section.**

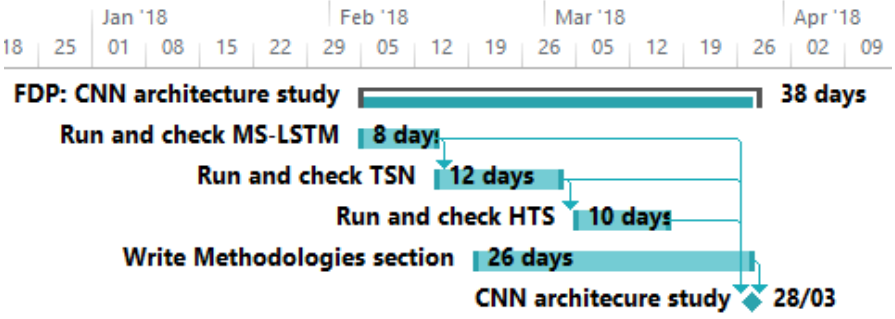


Figure 1.15: Task 3 Gantt diagram.

[1]Repository: https://github.com/harvitronix/five-video-classification-methods

After completion of this task, the following is **4. FDP: Speed-up proposal**, which begins the 29th of March and finishes the 11th of May. It is composed of:

- **4.1 GPU video decoding methods search.**
- **4.2 Convert HTS to PyTorch.**
- **4.3 TSN and NVVL system construction.**
- **4.4 Write Tools and Approaches sections.**

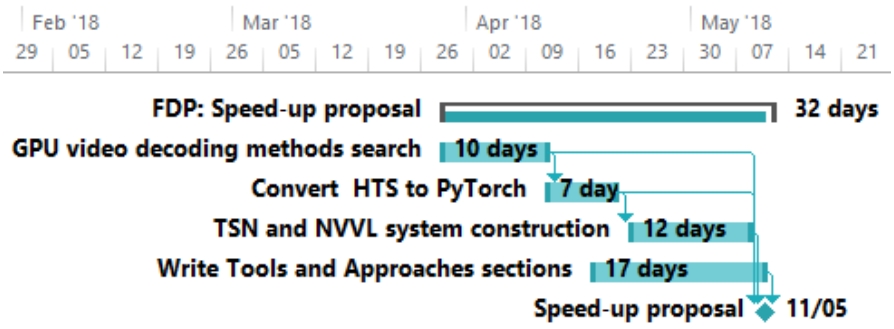


Figure 1.16: Task 4 Gantt diagram.

The last task is **5. FDP: Speed-up proposal**, beginning on May 14 and finishing the 6th of June. It is composed of:

- **5.1 Network training and evaluation.**
- **5.2 Results discussion.**
- **5.3 Write conclusions.**

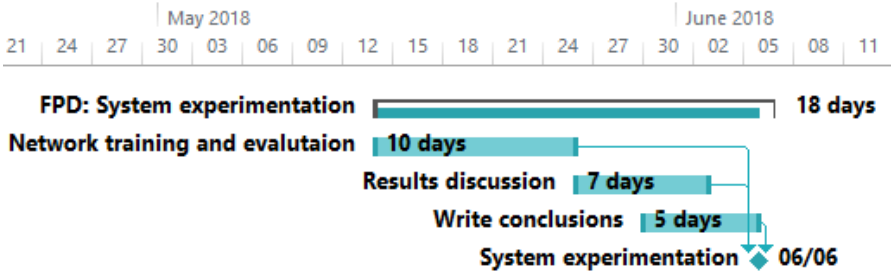


Figure 1.17: Task 5 Gantt diagram.

1.6 Outline

This document is structured as follows: Chapter 1 introduced the project and its motivations, gave a detailed description of the action recognition state of the art and outlined the main goals to accomplish and its distribution in the calendar. Chapter 2 describes the hardware and software tools that are going to be used, as well as the selected testing dataset and deep learning framework. Then, in Chapter 3, we review the most modern approaches, further detailing three outstanding ones. Following this, in Chapter 4 the acceleration proposals are presented, and two of them are tested. Finally in Chapter 5, the main conclusions of the Thesis are extracted, as well as future lines of research.

Chapter 2

Materials and Methods

In this second chapter we detail the different materials and methods we will use in the Thesis, mainly in software, data, and hardware. The chapter is divided in: Section 2.1, where the context of the material and methods is presented. Section 2.2, where three deep learning frameworks are studied and one is selected. Section 2.3, in which common action recognition datasets are introduced. Section 2.4, presenting the server where experiments are going to be run.

2.1 Introduction

For being able to realize this work, we need three types of resources. First, a programming framework that allows to increase the level of abstraction at the time of implementing deep learning models, training them and testing them. Although we could ignore it and program at a lower level, we would loss the focus of the main project task, as well as it would take from us an unnecessary large amount of time.

Second, we have to find suitable datasets for training models in the task action recognition. Moreover, we need one that is large enough, due to the nature of deep models and their easiness for overfitting with the training set.

Third, a powerful enough computer is needed for performing the multiple operations a deep learning system carries out. Also, it needs to have enough storage and memory for holding the used datasets.

2.2 Frameworks

With the raise of deep learning, easily writing code representations of models in a flexible and fast manner has become a trending necessity among researchers. For this reason, deep learning frameworks are nowadays needed. The list of them is hugely growing each year, and we need to properly focus the search on one that is adequate to our specific needs. Subsequently, we introduce three possible candidates: TensorFlow, Keras, and PyTorch.

2.2.1 TensorFlow

TensorFlow[19] is a library for numerical computation and using data flow graphs. Although actually, its most widespread use is the experimentation and development of deep learning models, it as well supports a wider range of machine learning algorithms, being this was the main purpose it was created for, by researchers and engineers at the Google Brain team. Its open-source nature (Apache 2.0 license) has contributed to ease the framework's improvement, thanks to the support of a large number of

contributors at its official repository[1]. TensorFlow, provides a flexible and fine-grained interface, implemented in both C++ and Python. By its architectural design, allows to scalably build and deploy systems seamlessly whether it is on Central Processing Unit (CPU) or GPU.

At the core of the framework lies the capacity of expressing computations as data flow graphs (nodes represent the operations while edges the data to be consumed). Several benefits emerge from this kind of representation: Easiness to detect and parallelize independent paths in the graph. Distribute operations across different devices and machines, for example, we could be training a model at the GPU, while performing data augmentation for the next batch on the CPU at the same time. Automatic differentiation, which allows to compute the derivatives of a function automatically. This method is based on the idea that any kind of operation, independently of its complexity, can be expressed as a group of primitive operations (add, multiply, divide...) and basic functions (exponent, root, logarithm...). Thus, by following the data flow graph scheme, each function operation can be broken down into these components, then, by applying the chain rule successively, the derivative can be obtained. In machine learning, this characteristic is heavily used at the time of finding gradients of loss functions on backpropagation based algorithms.

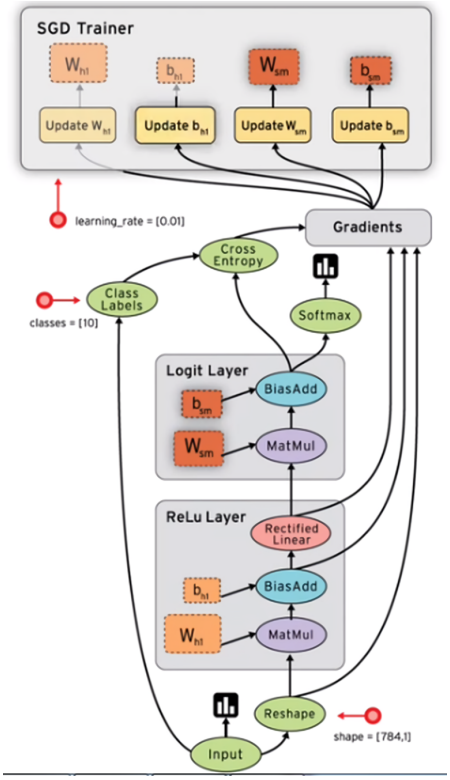


Figure 2.1: Computation graph of one single-layer network

[1] https://github.com/tensorflow/tensorflow

2.2.2 Keras

Keras[20] is a high-level deep learning framework fully written in Python developed by François Chollet. Aimed for fast prototyping and simplifying the construction process of neural networks, facilitating the entrance of beginners to the field. For accomplishing these objectives, it acts as a front-end for three deep learning libraries, Computational Network Toolkit (CNTK)[21], TensorFlow (which officially supports Keras) and Theano[22], allowing to interchange them easily and without almost any distinction, and being able to directly access them within Keras. Moreover the selection between GPU or CPU when executing the model is done, by default, in the background. Apart from the main repository[2], the community can extend the framework through new additions at the contributions repository[3] (both, under the Massachusetts Institute of Technology (MIT) license).

Keras allows to create models as a sequence of layers (Sequential Application Program Interface (API)) or as a graph of them (Functional API). The first approach is directed to creating simpler model by stacking layers linearly, obtaining a sole linear path from the input to the output. On the other hand, the latter approach enables to treat each layer as a node of a graph, being possible to have multiple entrances and exits on a model. Thus, enabling the design of more complex models without much little effort.

2.2.3 PyTorch

PyTorch[23] is both a Python tensor computation kit and a machine learning platform, mainly aimed for research. Its development has been mainly driven by Facebook Artificial Intelligence Research (FAIR) together with many contributors at GitHub[4]. Although being in early beta, it is enough mature to stand out within modern deep learning frameworks. As it may be appreciable, it is well influenced by Torch[24], a scientific computing Lua framework focused on supporting a wide range of machine learning algorithms. Specifically, PyTorch uses the same C back-ends as the ones implemented by Torch (TH, THC, THNN, THCUNN), thus obtaining a resulting performance on-par with Torch, or even better.

As a tensor kit, PyTorch offers a variated number of operations, quite similar to the ones NumPy has, being it proposed as a GPU alternative for it. Indeed, when transforming a tensor from PyTorch to NumPy, the converted version and the original will share the same memory location, meaning that if one is modified, the other will also be. Furthermore, it also incorporates automatic differentiation.

When used for machine learning, PyTorch has a differentiable characteristic, it operates with dynamic graphs, instead of the static versions used by a great number of deep learning frameworks. Concretely, this means that instead of defining a graph, compiling it and then being hardly difficult to change its logic at runtime, PyTorch creates a new graph on each model execution, allowing to vary the interns of it each time. This has several benefits, for example, in a network, the number and type of hidden layers could change depending on an epoch counter, flexible length sequences could be easily added for RNNs or even some parts of the network could be re-executed with different hyper-parameters until some layer outputs meet certain conditions such as

[2] https://github.com/keras-team/keras
[3] https://github.com/keras-team/keras-contrib
[4] https://github.com/pytorch/pytorch

expected loss or only a few number of dead neurons. This could also be the inverse, the execution of some layers could be skipped. For example, batch normalization could be avoided if an output is under a threshold mean and variance. Specifically, all this logic can be written with standard Python conditionals. Furthermore, the implementation of the models follows an object oriented approach, with design principles like inversion of control, where the network architecture is defined in a class and its parent handles all the underlying logic. These enhances extensibility and modularity. Also, layers can be added in a similar way than in Keras, following either, a sequential or functional approach.

2.2.4 Picking a Framework

Since the objectives of this work are mainly related to experimentation and research of possible action recognition pipeline improvements, we need a framework that is adaptable enough, easily supports fine-grained and coarse-grained tensor operations and layers, and offers high expressibility. For these reasons we select PyTorch as the framework we are going to work with. It presents both a high level API similar to Keras and a vast number of lower level operations. The approach it takes at the time of implementing networks as objects helps to understand and debug the model implementations. Furthermore, since it has obtained such a good reception between researchers since its first official release (at the beginning of 2017), many industries are giving PyTorch support to newly interesting tools, helping to create an ever stronger community.

Finally, although we have seen that it is efficient enough, we can better aim at production releases of PyTorch projects thanks to its incoming integration with Caffe2[1] (which since less than five months ago lives inside the PyTorch project) and 1.0 release, which will introduce more efficient operations at the same time it will make the model conversions much easier.

2.3 Datasets

In this section we introduce nine action recognition datasets as well as a summary table which highlights important information about each of them. The reviewed datasets have been selected in regard of their innovation in terms of how data is acquired and presented, their scale, which has to be large enough for properly training deep networks, and the types of actions they include.

2.3.1 RGBD-HuDaAct

[25] Motivated by the limited capacity of RGB methods at the time (2011) for the task of activity recognition (specifically in home environments with elderly people), authors present a dataset consisting of human daily actions recorded with both depth and RGB low cost cameras. Adding this new type of information (depth), enables to better capture the scene context as well as the motions itself, and thus reduce the recognition ambiguity that could be originated from the contemporary feature extraction processes.[1]

[1] https://caffe2.ai/blog/2018/05/02/Caffe2_PyTorch_1_0.html

More in detail, it consists of 12 classes and more than 1000 videos, giving a total of around 46 video hours. The classes cover actions ranging from *make a phone call* and *take off the jacket* to *mop the floor* and *eat meal* or *drink water*.

2.3.2 UTKinect-Action3D

[26] In order to test the new depth-based action recognition method developed, authors collect a dataset of RGB-D action sequences in indoor scenarios. Moreover, skeletal joints are extracted with the help of the recording Kinect. On the other hand, actions are captured from different angles and poses in order to demonstrate the view-invariance of their method.

Concretely, the dataset presents a collection of short videos taking from 1 to 4 seconds, with a mean of 20 videos per class in a total of 10 classes. The complete list of actions together with an example of it, can be observed in Figure 2.2.[2]

Figure 2.2: Sample frames for each action of UTKinect-Action3D, from left to right and top to bottom: *walk*, *stand up*, *sit down*, *pick up*, *carry*, *throw*, *push*, *pull*, *wave hands*, *clap hands*. For each row: Above RGB data, below RGB-D data. Figure extracted from [26].

2.3.3 UTKinect-FirstPerson

[27] Authors of this paper present two independent datasets recorded from the perspective of mobile robots. Concretely, the aim is to learn how people interact with them, since this information is of great interest at the time of developing new daily life applications that have these machines as its core, like surveillance or social care. Precisely, one robot has an humanoid appearance while the other does not. This is done in order to further explore the difference in behaviours that could arise from treating with each type of robot.

As an improvement from similar datasets of this type, depth data is also added to the RGB recordings for fully capturing the two type of motion information present in the dataset: (1) Ego-motion, related with the camera movement. In this case originated from both the autonomous movement and the interaction-resulting one, (2) independent motion, which is caused by the movement of subjects and objects in the scene.

Regarding the datasets organization, they present both the RGB and depth records as well as the skeletal joints of the subjects. They are divided into nine classes, each one

with specific actions. For the humanoid robot, we can see ones like *hug*, *point*, *punch*, or *shake hands*. On the other hand, for the non-humanoid robot they include: *ignore*, *pass by the robot*, *run away*, *stand up*, or *wave to the robot*.[3] A sample of some actions can be observed in Figure 2.3.

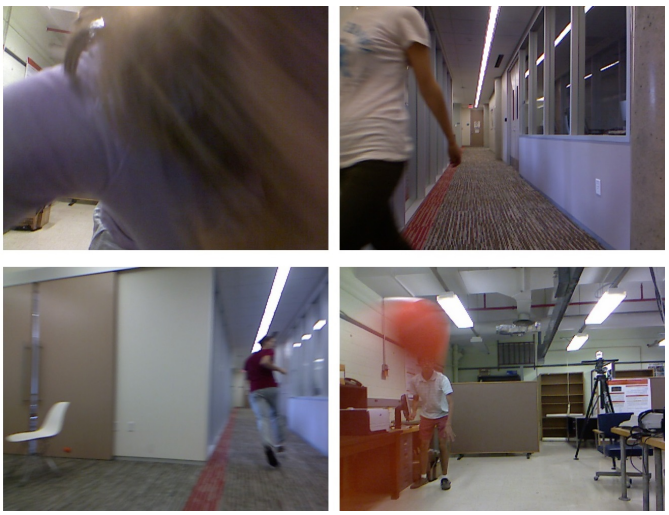


Figure 2.3: RGB frames of the UTKinect-FirstPerson dataset. Figure extracted from [27].

2.3.4 MHAD

[28] Since many datasets rise problems like being limited to some capture modalities and purposes, or not giving information about the hardware used for its creation (e.g, videos from YouTube), which could bias experiment results, authors propose a multimodal database of captured actions. Moreover, in both paper and project page, they give a detailed description of how it was captured and synchronized, together with the devices used.

In particular, data is acquired from five different sources: (1) Motion Capture (Mocap), which allowed to extract 3D movement and skeleton information; (2) cameras: two groups where used for stereo capture, while two more aimed for recording multi-view RGB data; (3) Kinects set in opposite directions (to avoid their pattern interference) for obtaining the depth data; (4) six accelerometers attached to the Mocap suit to obtain more precise temporal information of the actions. They were located at the ankles, wrists and hips; (5) audio through four microphones, since the combination of visual and auditive information has been proven to improve. One microphone is situated on the floor in order to obtain the audio that can be propagated through the action surface. The three other ones surrounded the motion scene. An example of the cameras and Mocap modalities can be seen in Figure 2.4.

The dataset is composed of 660 videos (with approximately 1.5 hours), having all the data sources synchronized after processing, and 11 action classes. They include: *jumping*, *bending*, *throwing a ball*, or *standing up*.[4]

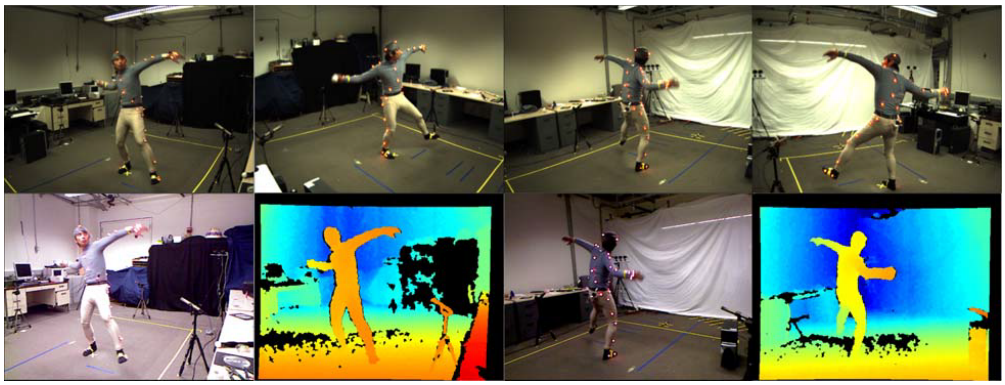


Figure 2.4: *Throwing a ball* class of the MHAD dataset, seen from each camera group and the two Kinects with both views: RGB and RGB-D. Figure extracted from [28].

2.3.5 ADL

[29] With medical applications in mind, authors present a first person view dataset of daily activity long-term videos. In fact, the label taxonomy was designed in terms of medical rehabilitation evaluations, in order to serve for future developments in the field. Another proposed application is to help in life-logging tasks for memory-loss patients.

Concerning the data, each video includes the object bounding boxes with their labels and its tracking, the interactions that happen in the video, positioning of the hands in the scene, and the activity classes. We can observe this annotations in Figure 2.5.

The dataset grows up to 10 hours of RGB video (more than a million frames) without scripted actions, in a division of 18 classes, including: *combing hair*, *make up*, *brushing teeth*, *laundry*, *washing dishes*, *making coffee*, *vacuuming* or *using cell*.[5]

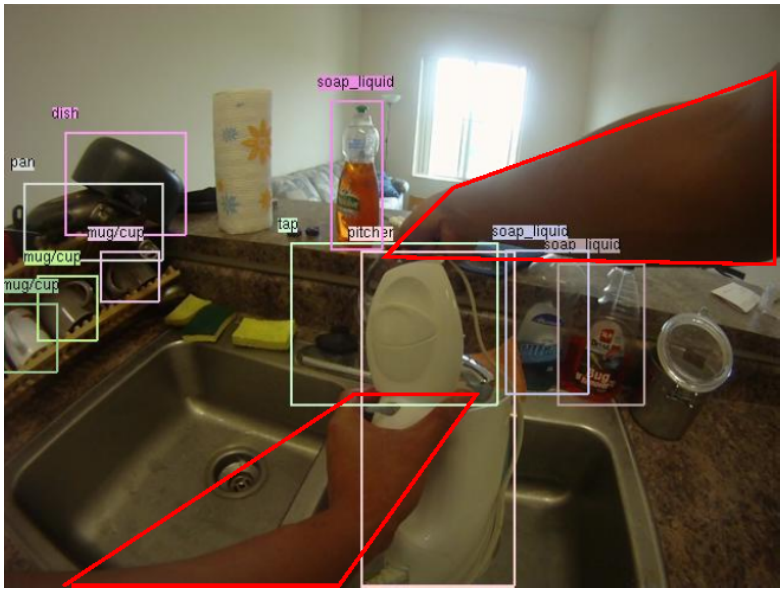


Figure 2.5: Sample ADL frame with multiple objects and arms detected. Figure extracted from [29].

2.3.6 NTU RGB+D

[30] Since many action datasets which include depth tend to have a small number of videos and classes, and with few actors, authors present a new large-scale RGB-D action dataset which covers more than 56000 videos corresponding to a total of 60 classes, differentiated in three large groups: daily actions, health, and mutual ones. Moreover, videos are recorded from 80 different positions with the Kinect v2 camera, being able to obtain infrared data apart from RGB-D images and skeleton joints. We can observe some of the actions in Figure 2.6

Regarding the actors, the videos present a total of 40 of them, whose ages range from 10 to 35. This solves another common issue on previous datasets, since a lack of actor diversity could affect intra-class variation.

On the other hand, they propose two types of model evaluation, cross-subject and cross-view evaluation. The former divides the subjects in half, obtaining two groups of 20 subjects. One for training and the other for testing. The later divides the different camera views also in training and test groups. For the first one, the front and side views are included, while for the second one, views are from left and right sides with an angle of 45 degrees.[6]

Figure 2.6: From left to right: RGB, RGB with joints, depth, depth with joints, and infrared versions of the same frame. Figure extracted from [30].

2.3.7 UCF101

[31] Given the limited number of RGB action datasets that included realistic scenes (without actors or prepared environments) and a wide range of classes until 2012, authors of this paper propose a new large-scale datasets of user-uploaded videos (YouTube). These present a much diverse type of challenges than the ones of previous datasets, since recordings can contain different lighting configurations, image quality degradation, cluttering, movement of the camera, and occluded scenes.

In regards to the size of the dataset, 13320 videos are divided into 101 classes that cover five action groups: Human-Human Interaction, Sports, Playing Musical Instruments, Human-Object Interaction, and Body-Motion only. The actions contained in the first and fourth groups can be observed in Figure 2.7.[7]

Furthermore, this dataset marked a milestone in what large scale action recognition datasets refers. It made possible to establish a well-known starting testbed to be improved as well as for benchmarking. Moreover, deep learning competitions where established around it, such as the different modalities of the THUMOS Challenge, which was run for three years in a row. After that, other large-scale datasets appeared, expanding the characteristics of the UCF101. For this, marking the start of an ever-growing number of diverse large-scale action recognition datasets, the UCF101 dataset is also worth of.

Figure 2.7: Classes for the Human-Object Interaction (blue) and Body-Motion Only (red) action groups from the UCF101 dataset. Figure extracted from [31].

2.3.8 STAIR Actions

[32] Centering its attention in the task of action recognition and understanding at home scenarios, and its possible applications like threat detection or health-care, authors present a large-scale daily action RGB video dataset with 100 daily-centered house actions grouped into five categories: object manipulation, washroom related, solo action, kitchen related, and multiplayer action. Each one can appear in 900 to 1200 videos, making a total of 102462 of them in the dataset. Regarding the video sources, either they have been downloaded from YouTube or provided by volunteers. Moreover they were labeled via crowdsourcing, with a program exclusively developed for this dataset.[8]

2.3.9 EPIC-Kitchens

[33] With similar intentions to the ones of the creators of the Activities of Daily Living (ADL)[29] dataset, the authors of this work present a large-scale RGB action dataset of first-person view kitchen actions, consisting of more than 11.5 millions of frames and 55 hours of recordings. These are divided into 39596 action segments pertaining to one of over 149 classes. On the other hand, each sample is annotated with bounding boxes of the objects used previous to, while, and after carrying out the action, obtaining a total of 454158 bounding boxes and 323 classes of objects. An example of the information contained in each video can be observed in Figure 2.8.

In terms of evaluation, three types of challenges are proposed: Action recognition, action anticipation, and object detection. For this, also two types of data splits are provided: The seen kitchens, where they appear in both training (80% approx.) and testing (20% approx.) sets. And the unseen kitchen, in this case the recordings of one kitchen appear only in a single set, training or testing. This split, although only accounts for around the 7% of frames in the dataset is challenging enough for modern deep learning models.[9]

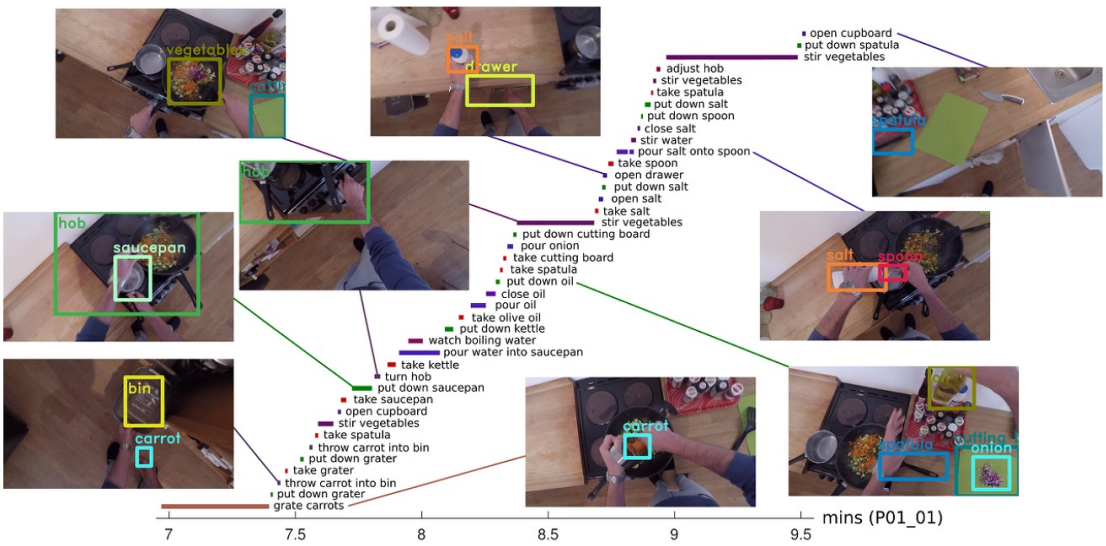


Figure 2.8: Subsequent actions with corresponding object detections. Figure extracted from [33].

2.3.10 Dataset summary

Here we present a summary table for the above datasets (see **??**). For each one, it contains the most useful information that we usually need for determining if a dataset suits our needs when coming to the action recognition problem. They are the following: The name of the dataset, the type of action the videos contain, the kind of data, whether if it is only in color or also includes depth, the number of classes and videos, the total amount of recording hours the dataset contains, the average number of videos per class, the Frames Per Second (FPS) at which the videos where captured, and the minimum, maximum and average length of each video in seconds.

Name	Action	Data	Classes	Videos	Hours	Vids./Class	FPS	Min s.	Max s.	Avg s.
UTK-FPH	Basic body	RGB-D	9	177	-	19	30	-	-	-
UTK-FPNH	Basic body	RGB-D	9	189	-	21	30	-	-	-
NTU RGB+D	General	RGB-D	60	56880	40	948	-	-	-	2,53
RGBD-HDA	Daily (indoor)	RGB-D	12	1189	46	99	30	30	150	139,28
UTK-A3D	Basic body	RGB-D	10	200	0,06	20	30	0,17	4	1,04
MHAD	Basic body	RGB-D	11	660	1,37	60	11	-	-	7,45
UCF101	General	RGB	101	13320	26,68	137	25	1,06	71,04	7,21
ADL	Daily (indoor)	RGB	18	-	10	-	30	-	-	99,50
STAIRS	Daily (indoor)	RGB	100	102462	156,54	1024	-	-	-	5,5
EPIC-K	Daily (kitchen)	RGB	149	39594	55	265	60	-	-	5,00

Table 2.1: Detailed summary table of the presented datasets.

[1]Project page: https://publish.illinois.edu/multimodalvisualanalytics/dataset/
[2]Project page: http://cvrc.ece.utexas.edu/KinectDatasets/HOJ3D.html
[3]Project page: http://cvrc.ece.utexas.edu/KinectDatasets/FirstPerson.html
[4]Project page: http://tele-immersion.citris-uc.org/berkeley_mhad
[5]Project page: https://www.csee.umbc.edu/~hpirsiav/papers/ADLdataset/
[6]Project page: http://rose1.ntu.edu.sg/datasets/actionrecognition.asp
[7]Project page: http://crcv.ucf.edu/data/UCF101.php
[8]Project page: https://stair-lab-cit.github.io/STAIR-actions-web/
[9]Project page: https://epic-kitchens.github.io/2018

2.4 Hardware

As one of the main key points for successfully obtaining high quality deep learning models is the need of vast amounts of data, the time taken for training the network with such quantities, can potentially become a bottleneck in the pipeline. To relieve from this, *Asimov* server was built and set-up for obtaining great performance and energy efficiency.

The complete configuration of the hardware is specified in Table 2.2. Mainly, the server relies on three NVIDIA GPUs for accomplishing the previously proposed goal. The Titan X is devoted to deep learning, whilst a Tesla K40 was also installed for scientific computation purposes thanks to its exceptional performance with double precision floating point numbers. In addition, a less powerful GT730 was included for visualization and offloading the Titan X from that burden.

With respect to the software, the server runs Ubuntu 16.04 LTS with Linux kernel $4.4.0-21$−generic for x86_64 architecture. The GPU are running on NVIDIA driver version 361.42 and CUDA 7.5.

Other relevant software includes Caffe RC3, PCL 1.8 (master branch `NUM`), Vtk 5.6, and Boost 1.58.0.1. All libraries and tools were compiled using GCC 5.3.1 and the CMake 3.5.1 environment with release settings for maximum optimization.

	Asimov
Motherboard	Asus X99-A [0]
	Intel X99 Chipset
	$4\times$ PCIe $3.0/2.0 \times 16(\times 16, \times 16/\times 16, \times 16/\times 16/\times 8)$
CPU	Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz [1]
	3.3 GHz (3.6 GHz Turbo Boost)
	6 cores (12 threads)
	140 W TDP
GPU (visualization)	NVIDIA GeForce GT730 [2]
	96 CUDA cores
	1024 MiB of DDR3 Video Memory
	PCIe 2.0
	49 W TDP
GPU (deep learning)	NVIDIA GeForce Titan X [3]
	3072 CUDA cores
	12 GiB of GDDR5 Video Memory
	PCIe 3.0
	250 W TDP
GPU (compute)	NVIDIA Tesla K40c [4]
	2880 CUDA cores
	12 GiB of GDDR5 Video Memory
	PCIe 3.0
	235 W TDP
RAM	4×8 GiB Kingston Hyper X DDR4 2666 MHz CL13
Storage (Data)	(RAID1) Seagate Barracuda 7200rpm 3TiB SATA III HDD [5]
Storage (OS)	Samsung 850 EVO 500GiB SATA III SSD [6]

Table 2.2: Hardware specifications of Asimov.

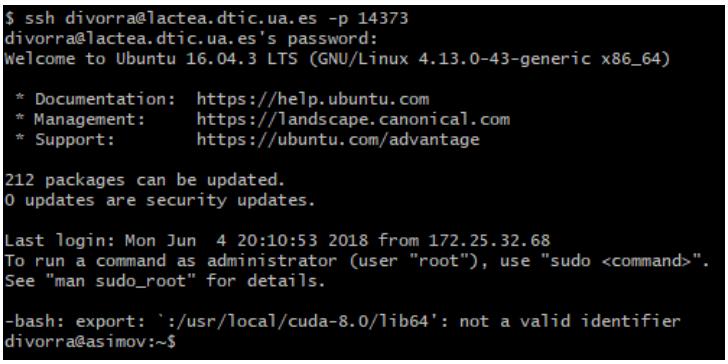


Figure 2.9: Asimov's SSH banner message and welcome screen with server info.

The server was configured for remote access using Secure Shell (SSH). The installed version is OpenSSH 7.2p2 with OpenSSL 1.0.2. Authentication based on public/private key pairs was configured so only authorized users can access the server through an SSH gateway with the possibility to forward X11 through the SSH connection for visualization purposes. Figure 2.9 shows an SSH connection to Asimov.

At last, a mirrored Redundant Array of Independent Disks (RAID)1 setup was configured with both Hard Disk Drives (HDDs) for optimized reading and redundancy to tolerate errors on a data partition to store all the needed datasets, intermediate results, and models. The Solid State Drive (SSD) was reserved for the operating system, swap, and caching.

2.4.1 Docker

Like in any kind of research, when conducting an experiment, an environment that is both deterministic and reproducible is needed, for that, the conditions in which it is run should be constant and known. In the field of computer science, this can be translated to first, obtaining the same expected output after each program execution, and second, that this happens with a pre-established set of system specifications and software versions. Moreover, is of our interest for the setting-up process, being simple and fast.

For achieving this purpose, several approaches exists. For example, in the case the program was written in Python, virtual environments could be used, being able to easily manage which libraries are installed in each environment and with which version (it can be done with *pip* or a manager like *Conda*). Although for other languages similar solutions exists, Conan[17] for C++, maven-env[18] for Java or use Python virtualenv for more than one language[19], these are either little mature or somewhat obscure.

[10] https://www.asus.com/Motherboards/X99A/specifications/

[11] http://ark.intel.com/products/82932/Intel-Core-i7-5820K-Processor-15M-Cache-up-to-3_60-GHz

[12] http://www.geforce.com/hardware/desktop-gpus/geforce-gt-730/specifications

[13] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x

[14] http://www.nvidia.es/content/PDF/kepler/Tesla-K40-PCIe-Passive-Board-Spec-BD-06902-001_v05.pdf

[15] http://www.seagate.com/es/es/internal-hard-drives/desktop-hard-drives/desktop-hdd/?sku=ST3000DM001

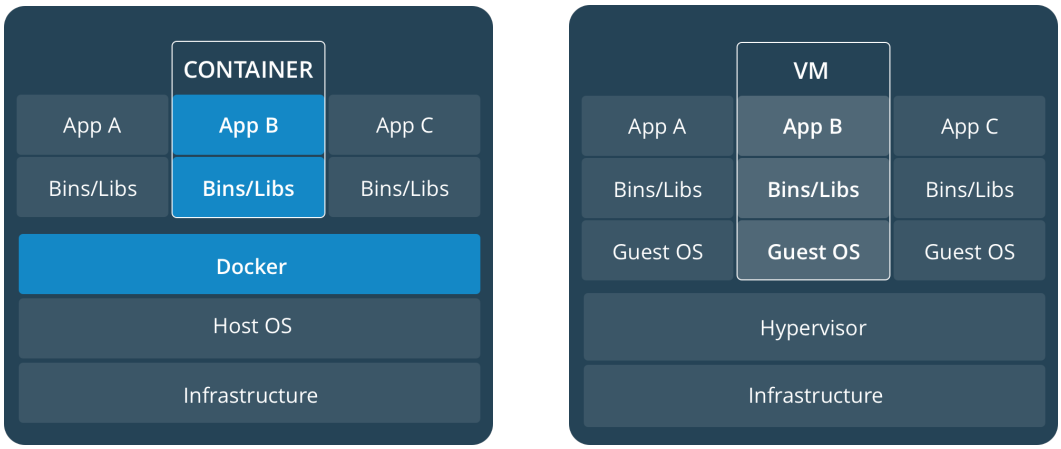
[16] http://www.samsung.com/us/computer/memory-storage/MZ-75E500B/AM

The question becomes more complex when experiment dependencies extend to more than one language and the the global environment is shared across different users, such as in a server host. In this case, the most straightforward solution could be to fully virtualize an operative system, install the needed dependencies and run our experiments in isolation from other users. A well know approach for these purposes, exist under the form of virtual machines. These are both, easy to use and create, but with the downside of having some performance overheads and lacking the ability to fully use all the machine potential.

One robust solution that meets all the requirements, follows the same virtualization lines and can make use of the available hardware without noticeably diminishing the performance is Docker[20].

This software is based in the concept of containers, which in contrast to virtual machines, where the virtual operative systems are managed by an hypervisor platform (Virtual Box, Parallels, Virtual PC...) on top of the host operative system, they run in the same space as any other program (user space), being able to access the kernel as these programs do, resulting in effective performance gains and start-up times (of seconds or milliseconds). As it can be understood, Docker is a container engine, offering a wide range of tools for building, managing and deploying them.

For creating and running each container, Docker utilizes images, which hold the description of the actual state of it. This can be analogically compared to classes (images) and object instances at runtime (containers). Furthermore, this concept can be extended to the means of inheritance, as each change in an image can be seen as a new layer added onto it, thus resulting in a new one (child class).



(a) Docker container. **(b)** Virtual machine.

Figure 2.10: Representation of abstraction layers on Docker and Virtual machines applications.

Indeed, this mechanism works similarly to how Git tracks changes, looking at the differences between the new file version and the past one, resulting in an appreciable disk space saving. Moreover, this derives into an abstraction where images can be managed using commands that result near to the ones of a control version system: Images can be both, pulled from other ones, extending their contents, and pushed, updating them. They are kept in either a local registry or a remote one, Docker Hub[21]. And even more, containers can be committed into the same image or into a new one. Also, the preserved changes in one of them can be rolled back. Finally, and greatly interesting for the kind of research machine learning conducts, the Docker engine officially supports the use of NVIDIA GPUs[22], being able to run models inside containers directly onto them, without apparent differences.

[17] http://blog.conan.io/2016/08/04/Conan-virtual-environments-Manage-your-C-and-C++-tools.html
[18] https://github.com/AlejandroRivera/maven-env
[19] http://jsatt.com/blog/virtualenvs-for-all/
[20] https://www.docker.com/
[21] https://hub.docker.com/
[22] https://github.com/NVIDIA/nvidia-docker

Chapter 3

Approaches

In this third chapter we examine the most recent works of action recognition carried out during the past three years. Setting in section 3.1 their context. Afterwards, in Section 3.2, important action recognition works related with deep learning are introduced. Then, in Section 3.3 we explain in detail the ones more suited for real time action recognition. We conclude this chapter with Section 3.4 with an explanation of the approaches selected for experimentation.

3.1 Introduction

In the following sections we review the most modern action recognition works carried out in the past three years. Mainly extracted from the most important computer vision and deep learning conferences: ICCV, ECCV, CVPR, NIPS, BMVC. Both non real-time and real-time systems are taken into consideration.

3.2 Overview: New action recognition approaches

In this section we briefly introduce modern notable works that that try to solve the task of action recognition. Although they not suitable for real-time applications, they are worth for taking a clever approach and their innovation. Moreover, for many of them, since their implementation is public, a link to the corresponding project pages has been added as a footnote.

First-Person Activity Forecasting with Online Inverse Reinforcement Learning

[34] This paper presents a novel method for predicting future behaviors by modeling the interactions between the subject, objects, and their environment, through a first person mounted camera. The system[7] makes use of online inverse reinforcement learning. Thus making it possible to continually discover new long-term goals and relationships.

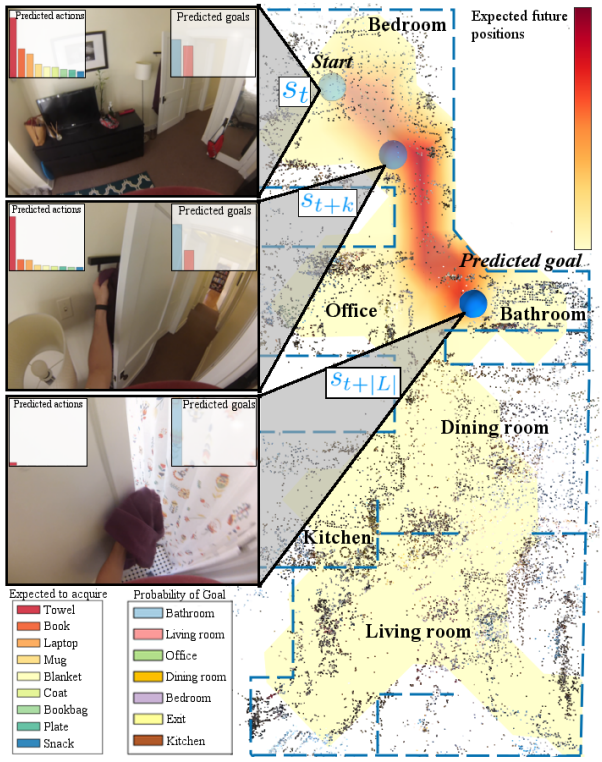


Figure 3.1: Environment map with possible future goals states (blue). Histograms show the probabilities of which object has been acquired (left) and the long-term goal (right).

Joint Prediction of Activity Labels and Starting Times in Untrimmed Videos

[35] By following a similar approach to that of the hybrid Siamese networks, this paper shows that is possible to simultaneously predict future activity labels and their starting time. It does so by taking advantage of features of previously seen activities and currently present objects in the scene.

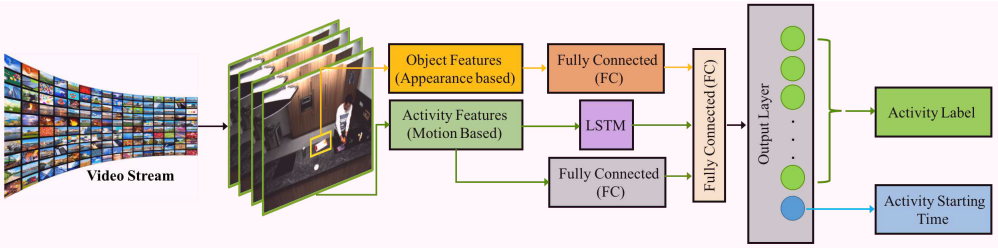


Figure 3.2: Pipeline representation of the proposed joint prediction approach.

Online Real-time Multiple Spatiotemporal Action Localisation and Prediction

[36] Thanks to the use of Single Shot multi-box Detectors (SSDs) CNNs, the system proposed in this paper is capable of predicting both action labels, and their corresponding bounding boxes in real-time (28FPS). Moreover, it can detect more than one action at the same time. All of this is accomplished by using RGB image features combined

with optical flow ones (with a decrease in the optical flow quality and global accuracy) extracted in real-time for the creation of multiple action tubes[8].

Deep Sequential Context Networks for Action Prediction

[37] In this case, for predicting action class labels before the action finishes, authors make use of features extracted from fully observed videos processed at train time, for filling out the missing information present in the incomplete videos to predict. Furthermore, thanks to this approach their model obtains a great speedup improvement when compared to similar methods.

Visual Forecasting by Imitating Dynamics in Natural Sequences

[38] A model that is capable of performing visual forecasting at different abstraction levels is presented in this paper. For example, the same model can be trained for future frame generation as well as for action anticipation. This is accomplished by following an inverse reinforcement learning approach. Also, the model is enforced to imitate natural visual sequences from pixel level.

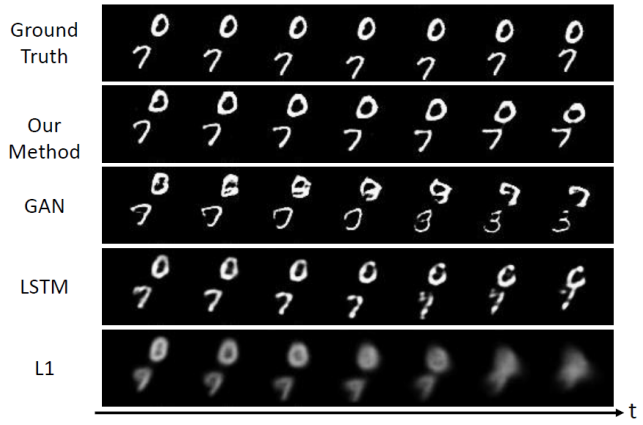


Figure 3.3: Example of predictions (second row) over the moving MNIST dataset.

Real-Time RGB-D Activity Prediction by Soft Regression

[39] The model developed in this paper is capable of predicting in real-time future activities labels on RGB-D videos. This is accomplished by making use of soft regression, for jointly learning both the predictor model and the soft labels. Moreover real-time performance (around 40FPS) is obtained by including a novel RGB-D feature named Local Accumulative Frame Feature (LAFF).

Temporal Convolutional Networks for Action Segmentation and Detection

[40] The authors present a new type of convolutional networks called Temporal Convolutional Networks (TCN) which are used for action detection and segmentation. By using a hierarchical convolutional layer approach, the model is capable of extracting long-range temporal patters.

Moreover, a TCN Encoder-Decoder system is built for performing the mentioned tasks. After training, it is able to surpass current similar approaches. Furthermore, the system presents a better performance than Bidirectional Long short-term memory networks (Bi-LSTMs) networks[9].

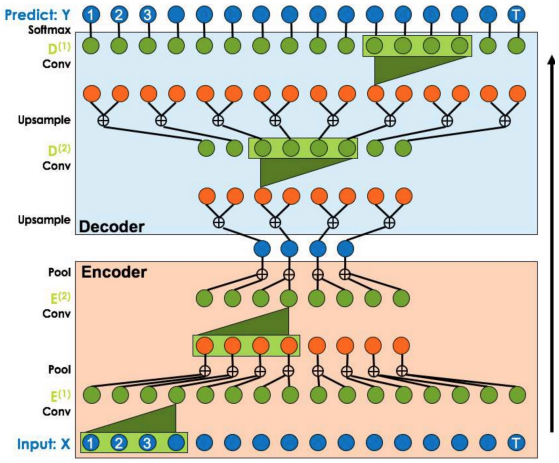


Figure 3.4: Diagram of the Encoder-Decoder architecture.

SST: Single-Stream Temporal Action Proposals

[41] In this paper, a system that is capable of presenting temporal action proposals on a video with only one forward pass is presented. Thus there is no need to create overlapped temporal sliding windows. Moreover, the system can work with long untrimmed videos of arbitrary length in a continued fashion. Finally, by combining the system with action classifiers, temporal action detection performance is increased.

This is achieved by extracting visual features with a 3-Dimensional Convolutional (C3D) network and feeding them to a Gated recurrent unit (GRU) layer which encodes the sequential information of the actions and outputs a fixed number of proposals at each timestep. The best proposals are selected by applying standard post-processing techniques[10].

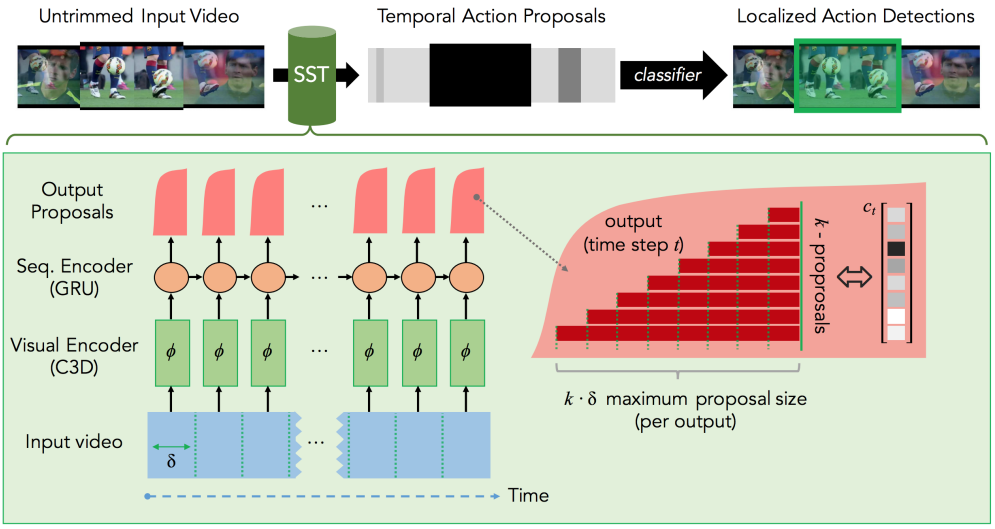


Figure 3.5: Representation of the developed system.

Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset

[42] Apart from reviewing current action classification techniques, authors of this paper present a new convolutional model, known as Two-Stream Inflated 3D convolutional neural network (I3D), which is used as a spatio-temporal feature extractor. After this, authors pre-train I3D based models on the Kinetics dataset, showing that with this approach, action classification performance on well known datasets is noticeably increased.

For achieving this results, different techniques are core in an I3D model. First, inflation of 2D convolutional modules into 3D ones (such as filters and pooling kernels). Generally, this is done by just converting NxN filters to its cubic form of NxNxN. Second, parameters pre-trained on ImageNet models are bootstrapped for being adapted to their own inflated versions, this is done by training the 3D versions on videos composed by the same single image repeated several times. Third and last, train two I3D systems independently, one with RGB videos and another with optical flow video features, and at test time, average their predictions[11].

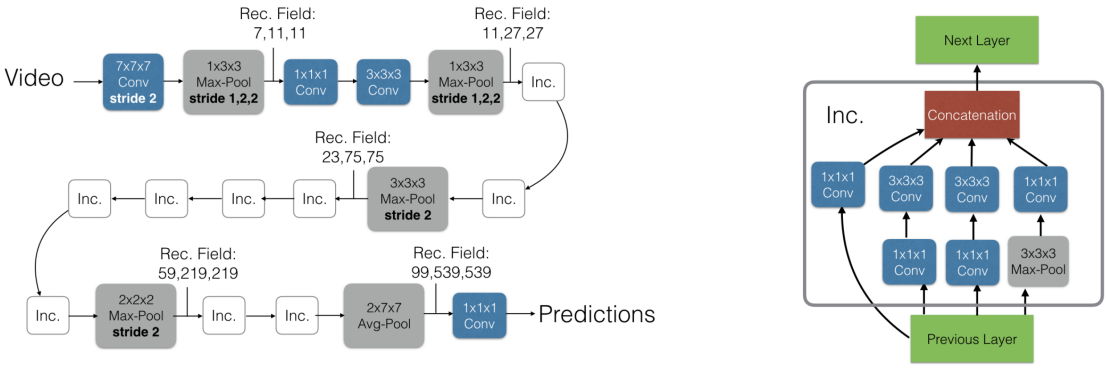


Figure 3.6: Inflated Inception-V1 architecture (left) and detailed Inflated Inception, *Inc.*, module (right).

Spatiotemporal Multiplier Networks for Video Action Recognition

[43] In this paper, a fully space-time convolutional two-stream network (named ST-ResNet) is proposed for the task of action recognition in videos. The first stream is fed with RGB data while the second, with optical flow features. The main particularity of this model is the existing interconnections between both streams. Moreover, for learning long-term relationships, identity mapping kernels are injected through the network. All of this allows the network to predict on a single forward pass.

Specifically, the network takes as base for both streams a ResNet-like architecture, in which features computed as output from a ResNet block of the motion stream (the one with optical flow features) are passed to the next ResNet block of the appearance (the RGB) stream, by multiplying it with the input of the RGB block. Thus creating a gating-like mechanism when the gradient is computed. On the other hand, for being able to support larger time intervals, 1D convolutions in conjunction with feature space transformations are applied through the network[12].

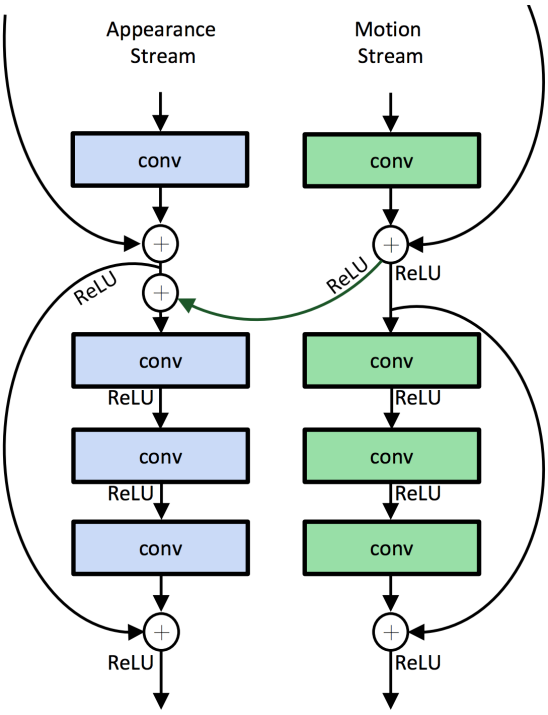


Figure 3.7: Detail of the modified ResNet block, with the gating moving from the motion to the appearance stream.

Predictive-Corrective Networks for Action Detection

[44] A new kind of recurrent neural network, known as predictive-corrective network, is presented in this paper. Authors make use of it for solving the problem of action detection in videos obtaining satisfactory results. In its basis the model: (1) Focuses on changes between frames, (2) predicts the future, (3) makes corrections upon it by observing what truly happens next.

Paying attention to variations, instead of to the actual frames, allows the model to reduce the number of computations done, as two subsequent frames could present high correlation between each other. Probably introducing needless repeated information.

In detail, the memory of the RNNs are the activations computed from the CNN layers one timestep before. Moreover, for allowing the network to correctly perform predictive-corrective computations, the memory of the RNNs is re-initialized every few frames (for example, every three frames). Furthermore, memory is also accessed to subtract it from the actual computed CNN activations. Comparisons with current methods show that predictive-corrective networks can compete with models based on two-stream approaches, without the need of making use of optical-flow features[13].

Asynchronous Temporal Fields for Action Recognition

[45] Authors of this paper propose a model that is capable of detailedly reason about aspects of an activity, i.e, for each frame the model is capable of predicting the current activity, its action and object, the scene, and the temporal progress. This is accomplished by making use of Conditional Random Fields (CRFs) that are fed by CNN feature extractors. Moreover, for being able to train this system in and end-to-end-manner, an asynchronous stochastic inference algorithm is developed.

Looking at the architecture of the model, it can be divided in two parts. First, the CNN extractor, which follows a dual stream approach, where RGB and optical flow features are computed, both using a Visual Geometry Group 16 (VGG16)-like structure. Second, the asynchronous temporal field (developed by the authors), which is in charge of time reasoning and predicting, making use of the features extracted by the two-stream network. Particularly, this is a fully connected CRF, structured for temporal modeling. Furthermore, for analyzing the results obtained with this architecture, a t-Distributed Stochastic Neighbor Embedding (t-SNE) embedding is computed for the intents of the predicted videos, showing that similar videos are clearly clustered together[14].

3.3 Real-time approaches

In this section we further detail three outstanding modern works that utilize deep learning for solving the question of action recognition in real time scenarios. Moreover, their code is also publicly available, and the corresponding links have been added as footnotes.

Temporal Segment Networks for Action Recognition in Videos

In this work [46], authors propose a CNN framework for the recognition of actions in videos, both trimmed an untrimmed. Along with it, a series of guidelines for properly initialize and operate such deep models for this task are proposed.

The framework aims to tackle four common limitations when using convolutional networks on videos. First, the difficulty of using long-range frame sequences, due to high computational and memory space costs, which can lead to miss important temporal information. Second, most of the systems focus on trimmed videos instead of untrimmed ones (several actions may happen in a video). Adapting to these would mean to properly localize actions and avoid background (irrelevant) video parts. Third, meanwhile deep models become complex, still many datasets are small in number of samples and diversity, lacking enough data for properly train them and avoid overfitting. Fourth, time consumed for optical-flow extraction can become a delay for both, using large-scale datasets and using the model on real-time applications.

[7]Project page: http://www.cs.cmu.edu/~nrhineha/darko.html
[8]GitHub repository: https://github.com/gurkirt/realtime-action-detection
[9]GitHub repository: https://github.com/colincsl/TemporalConvolutionalNetworks
[10]GitHub repository: https://github.com/shyamal-b/sst
[11]GitHub repository: https://github.com/deepmind/kinetics-i3d
[12]GitHub repository: https://github.com/feichtenhofer/st-resnet
[13]GitHub repository: https://github.com/achalddave/predictive-corrective
[14]GitHub repository: https://github.com/gsig/temporal-fields

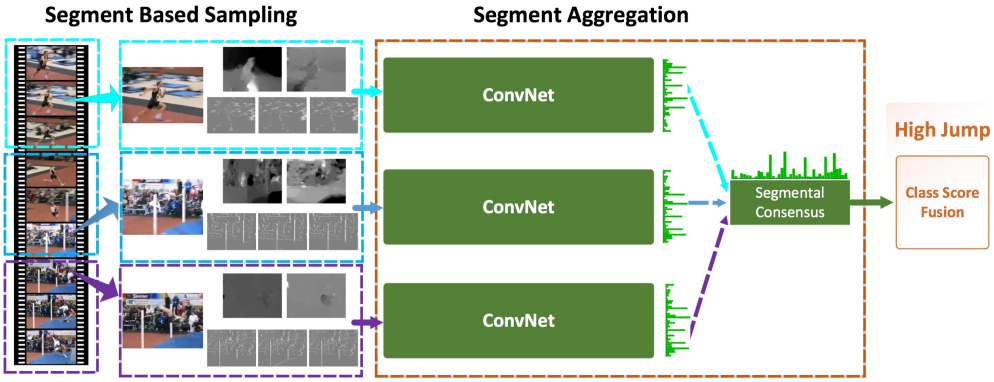


Figure 3.8: Representation of TSN framework. First a snippet is extracted from each of a fixed number of segments that equally divide the video, Then, features such as optical flow or RGB-diff (top and bottom images of the second process column) are extracted. After passing through the corresponding stream, an aggregation function joins the individual snippet class probabilities. Then, softmax is applied for obtaining the final video action class.

Tackling the first point, authors propose the use of a sparse temporal sampling instead of dense one, being able to avoid redundant information consecutive frames could carry. Specifically, the video is divided in a constant number part from which a random snippet (short frame sequence) is extracted. For each snippet the same set features are extracted (RGB, optical flow, RGB-diff) and passed to its corresponding CNN stream. After obtaining the predictions for each snippet, a selected global merging function (named by the authors as *segmental consensus function*) is applied over all the snippets for joining the partial predictions. Lastly, through softmax the final action class is computed. These functions are applied for each action class. They include either max pooling or average pooling between the class prediction values of all the snippets. Other two are: Top-K pooling, which combines the previous two functions by obtaining the average over the K greater class prediction values. And attention or linear weighting, in which both a weighted sum over the class prediction values is performed, having these weights being learned together with the network parameters.

Briefly, for the case of untrimmed videos (which diverges from the task of this work), they are divided in a fixed number of windows with the same duration in which the previous mentioned action recognition process is applied. Then, for obtaining the list of predicted actions taking place in the video, one of the two advanced functions (attention weighting or top-K pooling) that can reduce the importance of sequence background and focus on the action instances is applied. This approach served as a determining factor for obtaining the first place of the untrimmed videos category at ActivityNet Large Scale Activity Recognition Challenge 2016.

On the third aspect, the guidelines are introduced: (1) Cross-modality initialization, where pre-trained weights on the RGB model are transferred to other kind of streams (optical flow and RGB-diff) through a series of simple transformations; (2) partial batch normalization by freezing all this kind of layers values (mean and variance) except the first one on the pre-trained models for properly adapting to the different data distributions of the other streams. To further prevent overfitting, a high dropout ratio is applied after where the global pooling takes place; (3) data augmentation operations, in which apart from horizontal flipping and random cropping, two new ones are applied: Corner cropping, where only the center of the image or corner sections is presented to the network, for avoiding it to only pay attention on the first area. And second, modified a scale-jittering for the task action recognition.

With respect to implementation details, different architectures are used: Inception v2 and v3 or ResNet-152. Regarding the input of the temporal stream, three approaches are investigated: Optical flow, warped optical flow (both using height and width axes) aimed at reducing camera motion influence and focus on the action subject, and RGB-diff (of pixel intensities between two consecutive frames) which as indicated above, are intended to speed-up the motion representation at the same time they maintain a correct accuracy.

Different datasets such as HMDB-51 and ActivityNet are used during the experiments. In these, the results on the combinations mentioned temporal streams are presented, showing that although the use of warped optical flow derives on better accuracy than common optical flow, the time performance is greatly decreased being almost three times slower than the later (from 14 to 5 FPS). Finally, by using the RGB channel along the RGB-diff one, the network is able to predict in real-time (FPS > 30) at more than 330 FPS while maintaining a good quality performance ratio, helping this to solve the fourth previously mentioned limitation. Another investigated aspect is the number of segments used, resulting in 3 the minimal number for applying the framework (only one is understood as a common two-stream approach) and 7 the optimal one for capturing the temporal structure and avoid saturating (obtain the same accuracy with more segments) the network. Then, by using the RGB and common optical flow combination with 7 segments, the aggregation functions are tested leading to the conclusion that for trimmed videos average pooling works best while for untrimmed ones both advanced functions give similar results. Moreover, the best performing architecture in the experiments results to be Batch Normalization (BN) Inception.

For having a better insight on how the network represents action classes, some of them (pertaining to HMDB-51 and UCF101) are visualized in images by using a gradient ascent method (use an addition instead of a subtraction for updating the weights). When comparing the results, it can be seen that with TSN, model center more its attention in the human motion in contrast to common two-streams which context motion influences more.

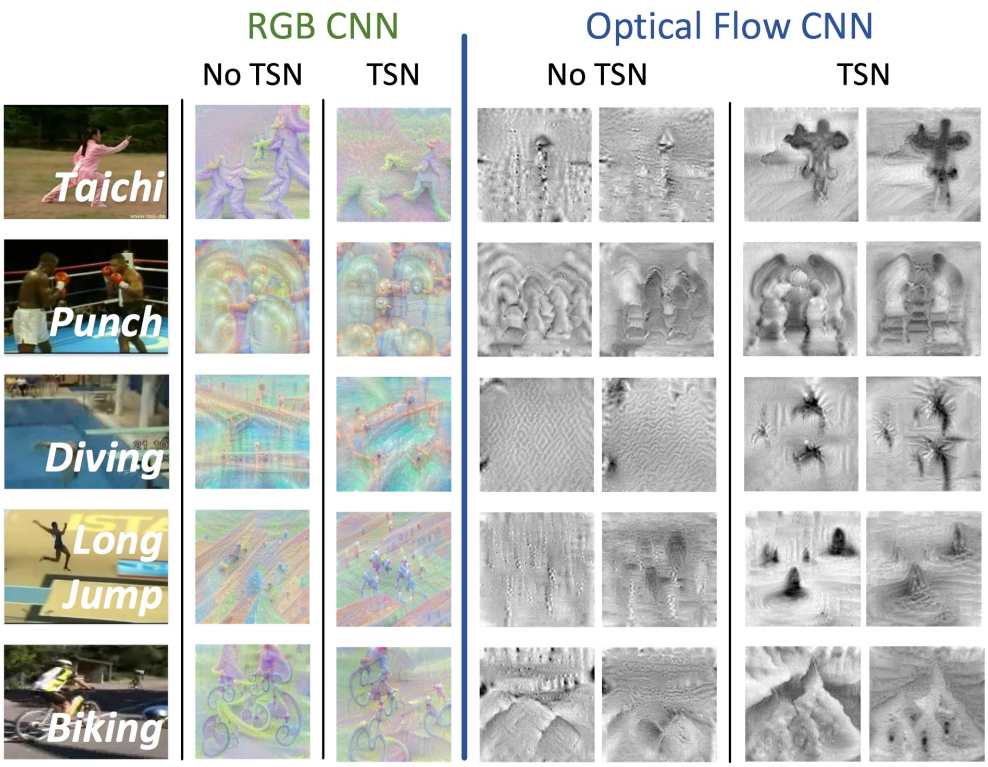


Figure 3.9: Visualization of five UCF101 class representations. It can be seen, for example in the diving class, that when using TSN more attention is paid to the swimmer (more of this action poses appear in the image) and less to the context motion for which the model lacking TSN encourages more, being possible to see that more waves appear on its image.

Encouraging LSTMs to Anticipate Actions Very Early

The authors of this paper [47] propose a multi-stage Long short-term memory network (LSTM) architecture combined with a novel loss function, that is capable of predicting action class labels in videos, even when only the first frames of the sequence have been shown. The model takes advantage of action-aware and context-aware features for succeeding in this task.

Specifically, the anticipation loss function penalizes false positives more severely as the sequence progresses, this is because some actions are only truly differentiable after the first few frames, e.g., swimming and diving. On the other hand, the contrary is done for false positives, since the objective is to assign the correct class label sooner rather than later.

The model bases the feature extraction process in CNNs while for interpreting the sequential information, uses LSTMs. For both context and action features, the same network, ImageNet pre-trained VGG16, is shared until one of its last convolution layers, where a stream for each feature appears.

For the context-aware, the rest of the VGG16 layers are kept until the last one where the number of outputs is adapted to the objective dataset (University of Texas Interaction (UT-Interaction), JHMDB, and UCF101). The feature vector is extracted from the

previous to the output dense layer, having the meaning of obtaining a representation of image scene.

For the action-aware stream, first Class Activation Maps (CAMs)[48] are extracted from the next convolutional layer, then after passing through dense layers, the feature vector is extracted in the same manner than the previous stream. In relation to CAMs, they as used for representing the image sections that contain more discriminative information about its class label. For this, they can indicate in which point of the image the action takes place.

Both streams are trained with still images before joining them to the LSTM model (being frozen in this phase), which uses as input the extracted feature vectors (context and action) for each frame of the sequence. Indeed, these is so since results where nearly the same and it required a smaller computation cost.

For the recurrent part, the context features are passed through a LSTM layer that diverges in two streams, the first is composed of a dropout function followed by an output softmax dense layer, where the anticipation loss is applied. The second, another LSTM which also takes as input the action features by concatenating them with the output of the first LSTM. Then, the same structure of the first stream is repeated in this one. Finally, the final loss is the result of adding the losses of both streams.

At time of predicting the correct action class, instead of obtaining a label for each frame (timestep) of the sequence, in order to make use of all their information, the final class is computed through average pooling of all the predictions up to the current frame.

On the results, the system can make correct predictions (over 80% of accuracy) with seeing less than 30 frames of each video. Moreover, experiments show that the combination of the two feature types provides a noticeable performance increase. Also, as action features are extracted from informative parts of still images they are more robust to unrelated movement than other motion type features. At the time of visualizing how the information is interpreted by the convolutional networks, for each stream (after splitting), is extracted an image resulting of combining through average pooling all the channels of the first convolutional layer. In this, action features centre its attention on the subject while context ones this happens through the scene.

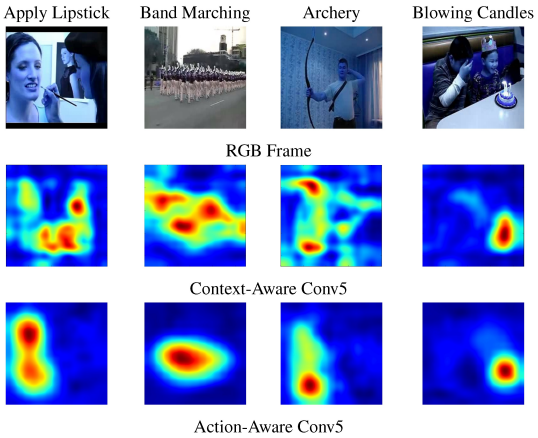


Figure 3.10: Representation of context (second row) and action features (third row). In the archery picture we can see that the second type of features are more active around the hand grabbing the bow while the first type, is focused on the other body parts e.g., the flexed arm or the bow shape.

Hidden Two-Stream Convolutional Networks for Action Recognition

Another side for approaching the question of real-time action recognition can be found in [49], where the use of a convolutional network for automatically compute optical flow is presented.

More in detail, in a first phase, a CNN denoted as MotionNet is trained unsupervisedly for the task of optical-flow estimation. After obtaining acceptable results similar to optimal traditional methods, the network is attached to a conventional CNN as the temporal stream part of the whole model, being the spatial stream similar in architecture SURE£ to the other one. Then, the network is trained (including MotionNet) on the task of action recognition from frame sequences. The approach enables the optical flow generator to be adapted to the characteristics of the task and further finding a suitable motion representation.

It is worth to mention that this approach is different from two other known methods that compute optical flow through CNNs both with supervised training. The first [50], uses as expected optical flow the one computed through handcrafted methods, setting the quality of the real representations as a complex approachable milestone. The second method [51], is partially restricted also by the training datasets, since it is synthetic, which although it possesses a true ground truth, due to this nature, it can be limited to specific datasets and tasks.

Regarding MotionNet, it is taken the assumption that if by applying the inverse warping function to the computed optical flow for two subsequent frames the first frame could be obtained, then the network would have perfectly approximated the optical flow function. Thus, it uses this comparison as one of the main contributors to the global loss function. For the architecture, a common convolution/transposed-convolution network with skip connections is trained from zero, with the particularity of using small kernel sizes (seem to better capture small motions like the variations in two subsequent frames (optical flow)), strided convolutions instead of max pooling since, as stated by the authors, for lowering the dimensionality of images on tasks regarding densely predicting pixels can have adverse effects. Although existing architectures like VGG16 or ResNet have been tested, this one obtains the best balance between accuracy and real-time performance. On the other hand, both the temporal and spatial streams use the VGG-16 architecture. Furthermore, results show that the model performs better than [51] while consuming much less memory space. Indeed, researchers implement a much less deep network, which although performs slightly worse than the original network, this time consumes around 70 times less. This can encourage to search for innovative architectures that are shallow rather than deep.

On the use of learned optical flow representation for action recognition, is found that still handcrafted features perform better (surpassing [51] and HTS). Finally, regarding the real-time aspect, different approaches are tested, leading to the conclusion that even when using the TSN framework, the network can still achieve it. Also, the previously mentioned reduced version with this framework can run at more than 480 FPS.

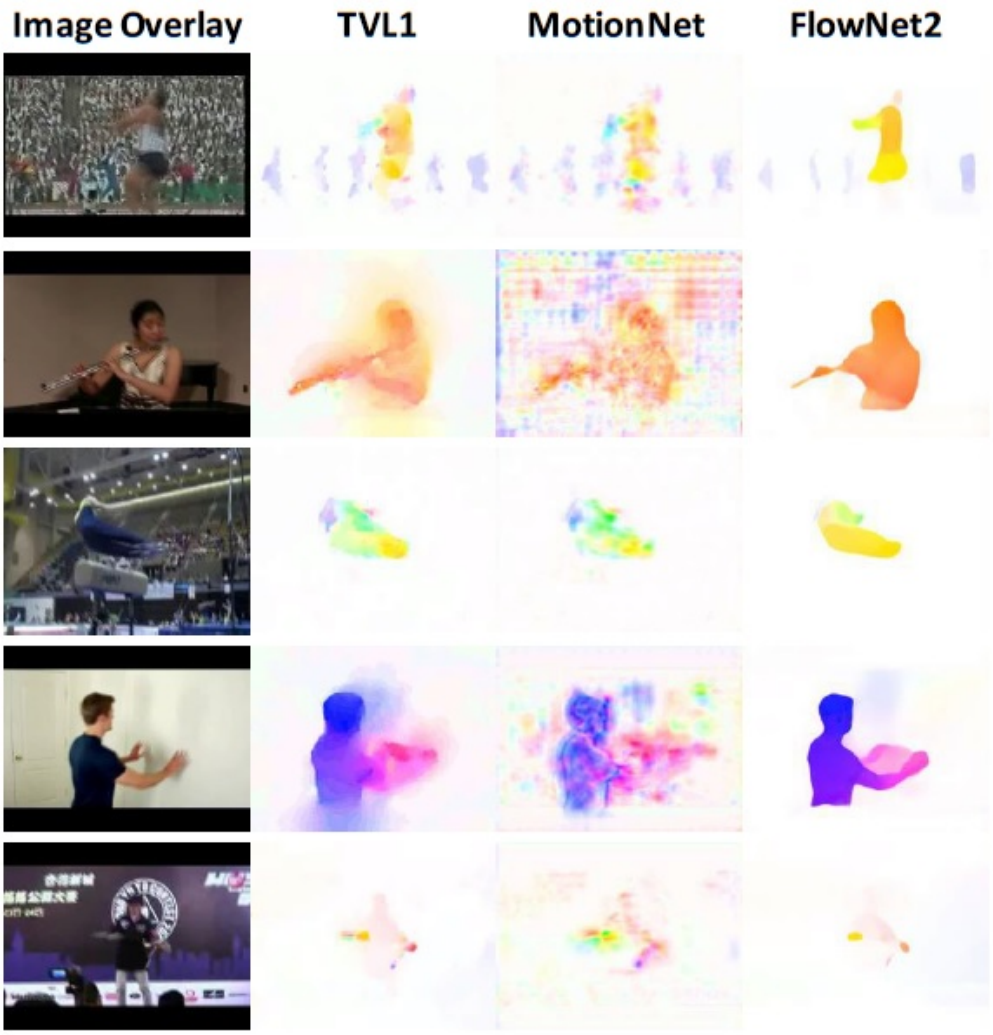


Figure 3.11: Different optical flow representations. While the first column method (handcrafted) performs better, the last column [51] extracts a more fine-grained version. Even with this, MotionNet still obtains a major performance while presenting noisy optical flow outputs. This, as authors indicate, can mean that the best motion representation still has to be found.

3.4 Selected approaches

Since we are aiming to improve the speed of a current action recognition system, we only focus our selection on the last three works review. Among them, we will discard [47], although the concept of frame anticipation is effective, the model requires more complex computations due to the two recurrent architectures used. Also, managing the hidden state of the RNN can become troublesome in the long run. For these, and since we need a lightweight model, we will focus the experiments carried out in the next chapter, on the TSN and HTS architectures.

Chapter 4

Experiments

In this fourth chapter we propose three methods for accelerating a video action recognition system. Furthermore, their validity is explored with different experiments. In Section 4.1, we briefly introduce the three proposals. Then, in Section 4.2 we study the possibility of GPU video decoding for accelerating the action recognition pipeline. Another proposal, infer optical flow with CNNs is explained in Section 4.3. In Section 4.4, we propose use the TSN framework as the third improvement. Moreover we combine it with the first one in order to show its viability. Finally, in Section 4.5 we summarize the experiment results.

4.1 Introduction

In the following sections, we present three approaches, which can also be combined, for improving the time performance of current action recognition systems, in order to orient them for real-time applications.

First, the use of TSN framework as the main network architecture, since it can provide us with high quality accuracy at inference time by using a small number of video frames. Secondly, for alleviate the I\O bottleneck that appears when working with large video datasets, we propose the use of video decoding applications through the GPU. For this we will make sure that we the frames do not see their image quality decreased. Third, extract optical flow by using CNNs for a more efficient and action recognition adapted implementation. Specifically, we will use the MotionNet architecture from HTS. Since it is implemented in Caffe, we will need to find a proper tool that allows use to convert the model into the chosen framework, PyTorch. Moreover, we will check the amount of precision loss that the conversion induces.

Finally, we will test the first and third proposals be training on a reduced set of the UCF101 dataset, and conclusions will be extracted.

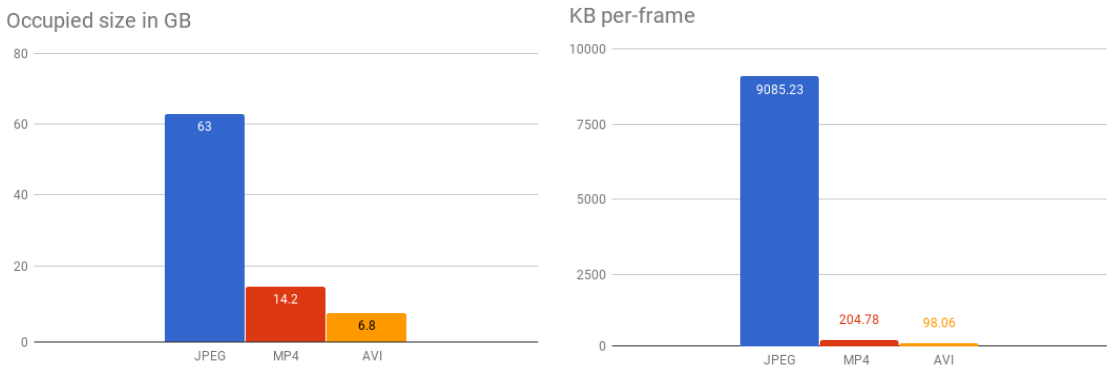
4.2 GPU Video Decoding

Since the beginning of the modern deep learning era, data storing and loading times have always been a bottleneck in the pipeline. Although recently we are witnessing great speedups thanks to new hardware technologies like SSD for storing, or data transferring devices (between CPU to GPU, and vice-versa) such as NVLINK, the issue persists.

Many of the research areas where this problem aggravates more are the ones which work with videos as the main dataset source. These include: predictive learning, video understanding, question answering, activity recognition, and super-resolution networks, between many others.

The main approach when tackling this problem in those areas is to first extract all the frames for each video of the dataset, for example by using FFmpeg[15], and save them in a high quality image format, rather than one with possible loosy compression and artifact generations, in order to properly train the network. This comes with an increasing need of storage space, since the more information willing to be kept, the larger in size our converted image dataset will be.

In the Figure 4.1 we can see the effects of storing the UCF101 dataset, composed of only 13320 videos in different formats. For the case of JPEG (image), it occupies 63 GiB, while in AVI format (video) it occupies 9.25 times less, 6.8 GiB. If it is transformed to the proper MP4 format, needed by NVVL (the video decoder we are going to use) with the corresponding number of frames, it occupies 14.2 GiB, still 4.44 times less. If we take this into a fine-grained level, such as frames, we can see that the storage differs by a large margin.



(a) Storage occupied in GiB.

(b) Mean size per frame.

Figure 4.1: Storage comparison between frames and video formats for the UCF101 dataset.

In order to alleviate this problem, a useful solution is to directly load video files into memory, decode the necessary frames, prepare them, and finally feed them to the network. Actually, APIs that can manage the first two steps exist: FFmpeg[16] libraries itself, and higher abstraction modules like PyAV[17] or PIMS[18], which both load data into the CPU. On the other hand, the (beta) Hwang[19] project, also supports NVIDIA GPUs through the use of their specific hardware decoder units.

Furthermore, those designed with machine learning tasks in mind, which can provide all the mentioned steps have been recently developed. Two are currently known: Lintel[20][52], and NVVL[21][53]. The first focuses on CPU loading (uses FFmpeg as backend), while the second targets GPU devices. Indeed, although being written in C++, it offers off-the-shelf PyTorch modules (dataset and loader). Moreover, another wrapper for CuPy[22] arrays has been created[23].

Regarding performance, we can see that it reduces by a large margin the I/O processing times, as it can be appreciated in Figure 4.2.

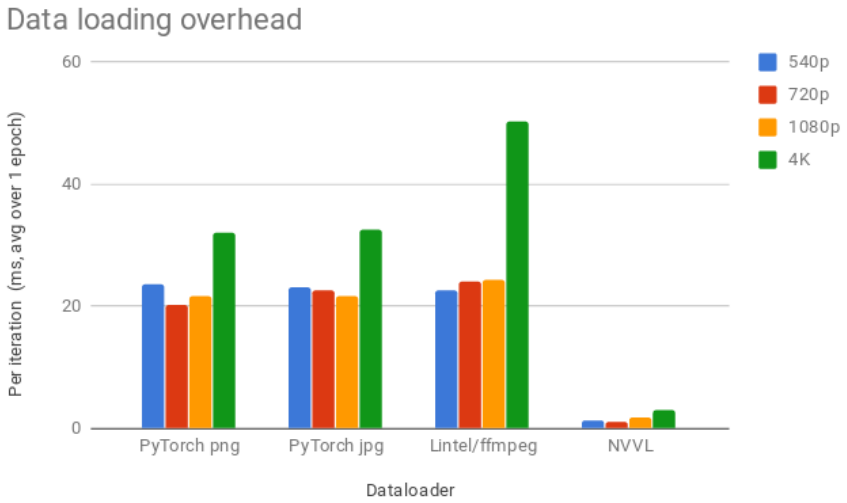


Figure 4.2: Average loading time (milliseconds) that 32-Floating Point PyTorch tensors take to be available in the GPU. The experiment was run on an NVIDIA V100 GPU over one epoch with batches of size 8. Figure extracted from [54].

More benchmarks that take into account memory usage and CPU loads can also be found in the blog post, while an even more detailed evaluation is located on GitHub[25].

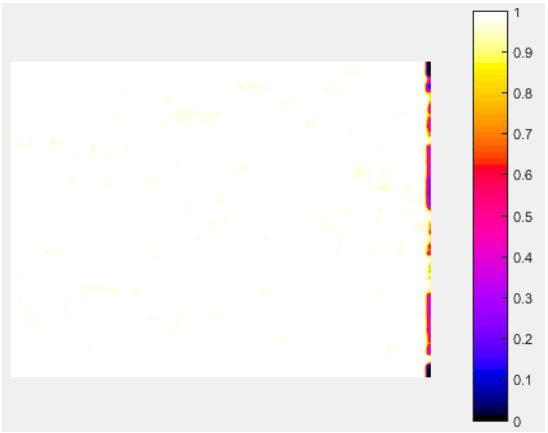
Regarding data, loading behaves like a sliding window of stride one, where frame sequences of a previously fixed length are subsequently loaded and returned as a single tensor. On the other hand, we can apply different transformations to these sequences: data type (float, half, or byte), width and height resizing and scaling, random cropping and flipping, normalizing, color space (RGB or Y: Luminance; Cb: Chrominance-Blue; and Cr: Chrominance-Red (YCbCr)), and frame index mapping.

Furthermore, we can compare the difference between the original frames and the ones loaded through NVVL. For this task, we are going to use the Structural Similarity (SSIM) index between two pictures, usually used in the video industry for measuring the visual difference we can perceive when comparing frames of an original and downsampled video. It ranges from 0 to 1, where 1 is given for two identical pictures and 0 for two completely different ones. For example, given the two frames obtained from the UCF dataset (*Diving* class) that can be observed in Figure 4.3, we can notice a green band on the right extreme of the NVVL loaded image.

[15] http://ffmpeg.org/
[16] https://github.com/FFmpeg/FFmpeg
[17] https://mikeboers.github.io/PyAV/
[18] http://soft-matter.github.io/pims/v0.4/video.html
[19] https://github.com/scanner-research/hwang
[20] https://github.com/dukebw/lintel
[21] https://github.com/NVIDIA/nvvl
[22] https://cupy.chainer.org/
[23] https://github.com/mitmul/pynvvl
[24] https://devblogs.nvidia.com/acclerate-machine-learnng-nvvl/
[25] https://github.com/NVIDIA/nvvl/tree/master/pytorch/test

Figure 4.3: Original frame (left) and NVVL obtained frame (right). The frames pertain to a sample of the *Diving* class in the UCF101 dataset

Apart from this, we can not perceive any other substantial degradation in quality. Indeed, the SSIM obtained is 0.992, indicating that this artifact is probably due to a bug rather than a low quality video processor. For reassuring this fact, we can compute the SSIM heat map, in order to locate other possible missed artifacts:



(a) Overview.

(b) Green band zoom in.

Figure 4.4: Heat map of above frames, the lighter the color, the closer to the original frame each pixel is.

Thus we can assume that there will be no harm at the time of incorporating this tool in a neural network training pipeline.

Now, we should pay attention to knowing which is the current time speedup we can obtain from substituting the image loading system of the TSN framework by a NVVL pipeline. For this, after adapting the frame-index generation functions, and integrating the video loader into it, we can perform the following:

1. Obtain a list of videos, get the total number of videos and the mean number of frames per video.
2. Extract all the frames from the videos, also convert them into the required NVVL video format.
3. Select the number of frames per video that are going to be loaded. For NVVL, all the frames have to be loaded.
4. Measure how much time it takes for extracting the told number of frames (into the GPU) on each occasion. For NVVL this only needs to be done once.
5. Obtain mean times and trend for the previous process and compare the results.

Step 1.

We will use the first 450 videos of the UCF split-1 train list obtained with the data tools provided by the TSN framework. This list is formated with a row for each video. In each row, the path to the video, the number of frames the video has and its class index. Due to this, the total number of frames can be obtained just by summing the second element of each row over the whole list. The resulting number is 87501. So the mean number of frames in a video is $\approx$194.

Step 2.

For completing this step, we can simply follow the instructions and commands provided in the repositories of each project. We have to take into account that the extraction process can take a quite greater amount of time than the video converting process.

Step 3.

In this case, we are going to load even number of frames, starting from 3 and finishing in 25, a total of 12 different instances. This has been selected since the authors of TSN test the model with 3, 5, 7, and 9 frames per video.

Step 4.

For obtaining an accurate measurement, we are going to repeat each execution 29 times. For computing the time we have used Python *time.time* function. Also, in order to free all the resources in each run, we are going to loop inside a bash script instead of inside the Python executing script itself, thus having the process killed automatically.

Step 5.

In this step we computed the mean values for each number of frames. The time taken for loading all the videos with NVVL is ≈24.18 seconds. On the other hand, we can plot the results obtained from loading sole frames:

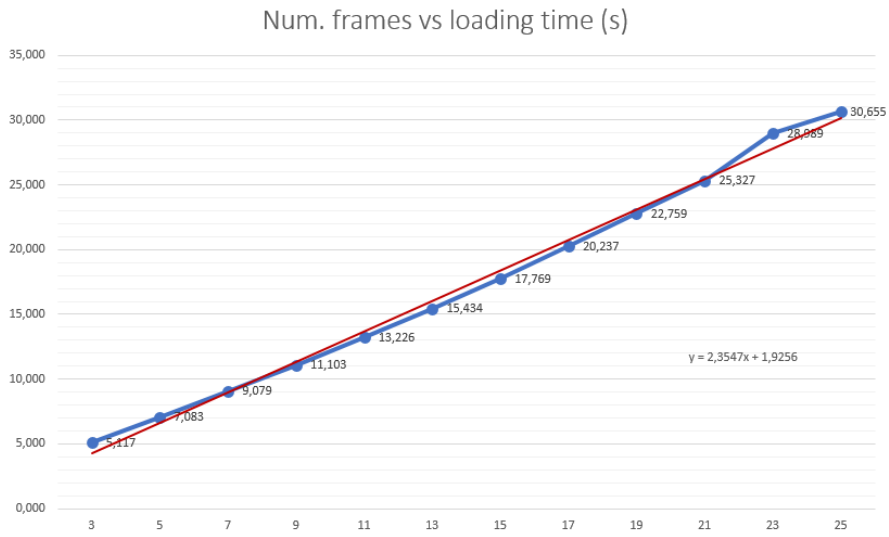


Figure 4.5: Mean loading time in seconds of each number of frames executed (blue). Trend line of from the obtained data (red). Y axis represents the loading time in seconds, while the X axis shows the number of frames used.

We can notice that the trend follows a lineal growth with respect to the number of frames loaded. Since we computed the equation defining the trend line (shown in the lower-right part of Figure 4.5), we can obtain a more precise approximation of the speedup achieved when using NVVL. For this, since we know the number of frames loaded with NVVL is the same as the mean number of frames obtained in **Step 1**, we just need to substitute it in the equation (X variable), obtaining a mean value of ≈458.74 seconds or 7.65 minutes. We have achieved an improvement on loading time performance, leading to a 18.97 times speedup when using NVVL.

4.3 Converting Caffe networks into PyTorch

In order to be able to use the pre-trained Caffe MotionNet in PyTorch, we need to convert both the model whose format is *.prototxt* and its weights (*.caffemodel*) to a PyTorch understandable version. Usually this is not a simple task, since both frameworks do not share neither the same model representations nor the file saving format.

A quite popular solution for solving this problem is the use of model converters, i.e, programs that alleviate the work load, and commonly with a few clicks and commands allow us to transfer a network in a source framework to a destination one. Two characteristics of these tools have to be taken into account: (1) Most of the converters are community-driven, (2) due to precision aspects (e.g. floating point arithmetic), the same experiments, but run on the destination network, can vary their output (rarely by a significant amount).

Regarding the first point, this leads to a situation similar to the one of high-level and assembly languages in the past, when intermediate representations did not exist and programmers created compilers exclusively for bridging the gap only between one pair, thus creating a confusing and vast amount of them. Currently, this is happening again, since the number of new and useful deep learning frameworks is constantly rising. Indeed, projects that keep track of the most effective converters do exist, one of the most popular ones is "Deep learning model convertor", located at GitHub[26].

On the other hand, current projects aimed at creating a standardized intermediate representation (an abstraction of this concept can be observed in Figure 4.6), backed by well known organizations are coming into existence. Specifically, two of them stand out of the crowd: MMdnn[27] and Open Neural Network Exchange (ONNX)[28]. The first one compiles a set of tools for conversion, model visualization, model compatibility testing, and code fragments for training and predicting generation. The second, apart from conversion and model visualization, also focuses on providing hardware optimization at time of exporting to its intermediary representation format.

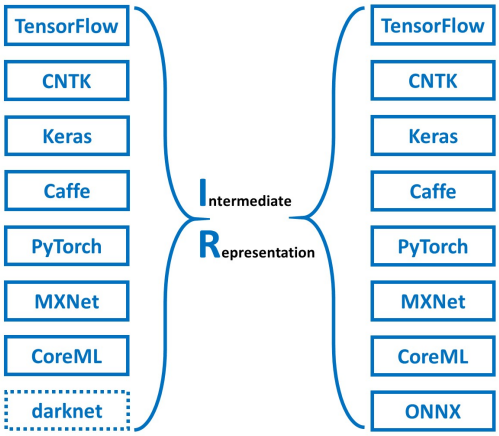
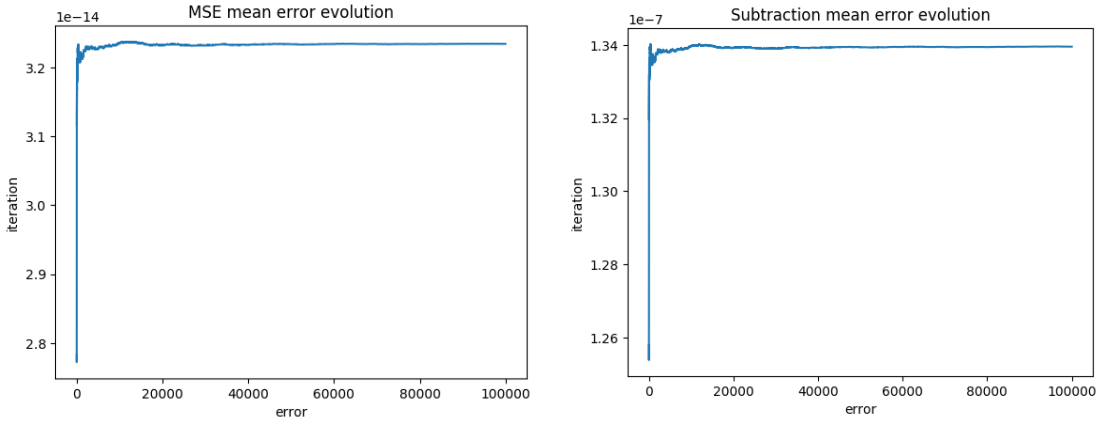


Figure 4.6: Deep learning frameworks interoperability concept (from MMdnn).

At the moment, ONNX only allows model exporting for PyTorch, and MMdnn although supporting it for Caffe, fails to properly convert the MotionNet model. Therewith, a pair Caffe to PyTorch converter has been chosen. Concretely, it first started as a set of Darknet You Only Look Once (YOLO) converters[29] (to and from Darknet, Caffe, and PyTorch). Later, the Caffe to PyTorch converter was maintained as a standalone project named "caffe2pytorch"[30].

This tool has the particularity of converting the model at runtime, instead of generating the model and network files in the destination save format. It does so by looping over the *.prototxt* file, reading the layers and creating the corresponding PyTorch versions of each one, keeping the same parameters or using the default ones if the equivalent layers are absent. Then, it proceeds similarly with the weights, through the use of the Numpy library.

For checking the error this converter can induce into the network, two metrics are used: Mean Squared Error (MSE) and absolute subtraction error. They are used for comparing how much the outputs of the original and converted network differ. For this, random inputs are fed to both networks in a 100,000 iterations loop. For each of these repetitions, the mean is computed and plotted in a graph, which can be observed in Figure 4.7. At the end, a minimal error is obtained. Being ≈3.23e-14 for MSE and ≈1.34e-07 for the absolute subtraction error.



(a) MSE measure.

(b) Absolute subtraction measure.

Figure 4.7: As we can see, both measures follow the same trend, stabilizing around the iteration 20,000.

Therefore, we successfully managed to convert MotionNet Caffe model into its PyTorch version assuring that no precision loss happened.

4.4 Proof of concept: Training RGB TSN with NVVL

So far we have shown how useful incorporating NVVL into a video-consuming deep learning pipeline can be, it allows us to reduce both the storage and data transfer costs at the same time we do not suffer degradation in image quality. Now, what only remains is to incorporate this tool into a common action recognition scenario, where we train and test a network for learning to categorize human actions.

Such a network is going to be TSN, since it has demonstrated a superior performance in the task at hand. Moreover, we propose to make use of the converted HTS Caffe model and weights, in order to avoid pre-computing the optical flow and being able to use NVVL also in this stream, focusing the resulting pipeline for real-time applications. In spite of the dataset we are going to use, memory limitations detailed below, and time constrains, we are going to focus the following experiment only for the RGB stream.

Before starting, we need to prepare the data into a format that is compatible with NVVL. As referred in the GitHub repository (link above), we need videos with either H.264 or HEVC (H.265) encoder-decoders (codecs), and yuv420p pixel format, also they can be in any container that FFmpeg parser supports.

[26] https://github.com/ysh329/deep-learning-model-convertor
[27] https://github.com/Microsoft/MMdnn
[28] https://github.com/onnx
[29] https://github.com/marvis/pytorch-caffe-darknet-convert
[30] https://github.com/marvis/pytorch-caffe

Moreover, we have to take into account the number of keyframes each video will have, i.e., a codec only contains a subset of all the frames that we see in a video, these are the keyframes. At the time of decoding, the rest of the frames are obtained by algorithmically inferring them through the keyframes. For this reason, when loading sequences that can start and end at any frame (similar to what we can do with NVVL), the system has to seek the nearest keyframe, which can be far before or after the starting frame. This can result into an underperformant execution, and for this reason when converting the videos, we have to indicate the frequency of keyframes per frame we want to have.

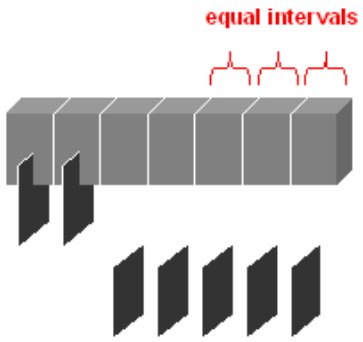


Figure 4.8: Representation of how keyframes can be evenly inserted into a video stream.

Developers of the video loader suggest to set one keyframe in intervals that correspond to the length of the sequences we are going to load. For example, if we are going to load sequences of length 7, then every 7 frames there will be a keyframe. Furthermore, they also provide the required commands to carry out this conversion with FFmpeg.

For our case, we are going to set every frame in the video to be a keyframe, this is due to the fact that currently the PyTorch wrapper (the C++ API seems more flexible) is intended for loading multiple frame sequences for each video with a sliding window approach of a fixed length. Although this length could be equal to the number of frames in the video, thus loading only a sequence per video, this would only work if all the videos had the same length, since this parameter, the sequence length, is global for the whole dataset.

For iterating over the dataset, we are going to use the data loader provided by NVVL PyTorch wrapper, where in each iteration it will load a batch of frame sequences. Since now, each frame is a sequence of length one, we need to set the batch size also to one. In this way we can easily know when the loader has fully output a video, add it to a list, and when we have enough videos, group them in a batch of the size we want for providing it to the network. Furthermore, for accomplishing this we also need to set to false the shuffle option in the loader.

Although we are ready for training our network, an impediment arises at the time of writing this work. Whether the videos have not been properly converted, or there is a code issue, the data loader seems to get silently stuck when loading some videos (more on these here[31]). For solving this, one circumvention is to create a loader for each

video instead of having one for the whole dataset.

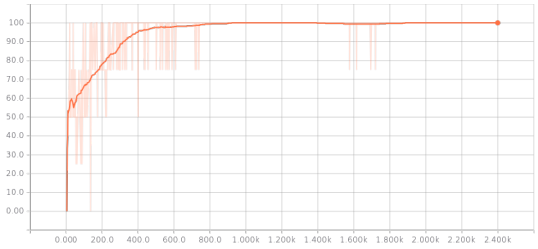
So far this works, but what happens next is that GPU memory is not properly freed, thus limiting the size of our dataset to the space available on the graphic card at the moment. For Asimov, this concurs in having around 240 videos for training and 160 for validation (only the Titan card supports NVVL).

For this, following the same lines of motivation proposed at the beginning of the document, we are going to select daily actions for the reduced dataset we can work with. Specifically, it is composed of eight classes from the Human-Object Interaction group of the UCF101 dataset: *Apply Eye Makeup*, *Apply Lipstick*, *Blow Dry Hair*, *Brushing Teeth*, *Cutting In Kitchen*, *Mopping floor*, *Shaving Beard*, and *Typing*. The training set contains 30 videos for each action, while the validation one has 20 of them.

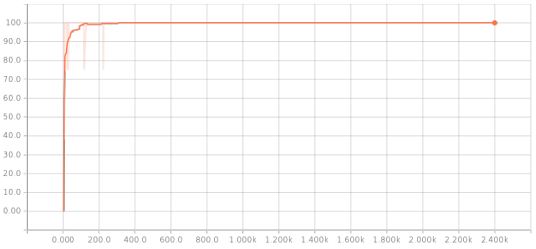
Regarding the training hyper-parameters, we are going to use the ones set by default for the TSN with the only exception of the batch size and number of epochs. For the former, we have set it to 4 due to the limited memory, for the later, we will perform 40 epochs, which is enough for the model to converge with this dataset. For the metrics, we will keep track of the loss and top-1 and top-5 accuracies for both the training and validation sets.

Once all the epochs have been completed, we finish our training with in a common situation, 100% of top-1 (and top-5) accuracy and zero loss. Clearly, our network has overfitted. This is due to the scarce amount of data and its limited variability. Moreover, such deep networks (BN Inception) are proner to overfit, since they have more flexibility (more number of parameters) for adjusting to the data they are consuming while in training. However, we obtain validation accuracies of 76.25% and 98.125% for the top-1 and top-5 versions respectively, with a loss of ≈1.52. Taking into account the data we are working with, these results are quite promising in comparison to what has happened in the training phase.

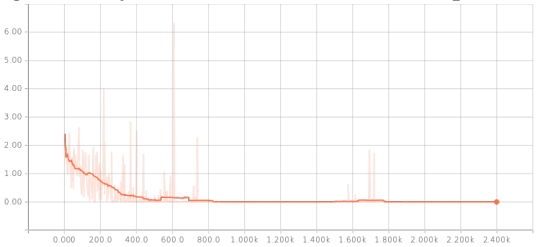
Now, we can get more insights if looking on how the training and validation have evolved. In the figure 4.9:



(a) Top-1 training accuracy.



(b) Top-5 training accuracy.

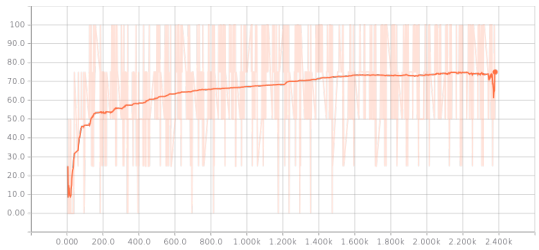


(c) Training loss.

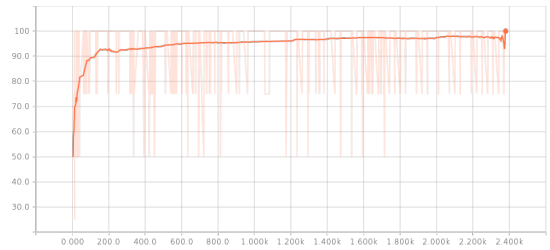
Figure 4.9: Training curves for 40 epochs, 60 iterations per epoch.

[31] https://github.com/NVIDIA/nvvl/issues/29

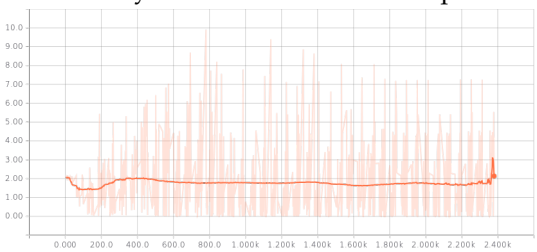
Here we can take note of two facts. First, the top-5 accuracy converges much faster (iter. ≈200) than the top-1 accuracy (iter. ≈800). Clearly, this is something that can bee foreseen, since it takes more time to learn the label of a video rather than guess it among five samples. Secondly, we see that the unsmoothed curve (shaded red) bounces between higher and lower values (accuracy and loss) among the first iterations. This effect can be better seen in the validation curves:



(a) Top-1 validation accuracy.



(b) Top-5 validation accuracy.



(c) Validation loss.

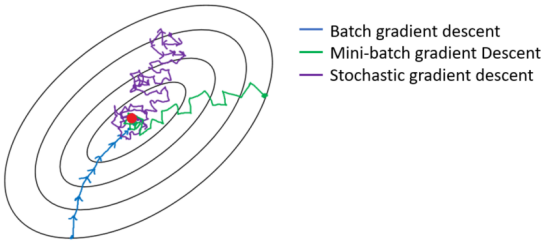
Figure 4.10: Validation curves for 40 epochs, 60 iterations per epoch.

This happens as a consequence of the small batch size we have previously set. The smaller is the batch size, the more number of weight updates we will perform. If it is to small, we could find the following:

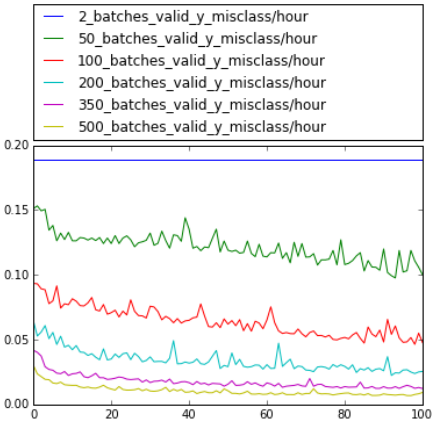
- Instability: The frequent updates will cause the metrics to wander, going continually up and down.
- Not meaningful updates: The reduced number of samples makes it to contain less information about the error (negative gradient) direction in each update, thus needing a major number of epochs for converging into the same accuracy than with a bigger batch size. This can be summarized as longer training times.
- Hit a local minima: Also known as plateau, and commonly induced by the previous statements, a small batch size can make that the network gets stuck on a non-optimum (nor sub-optimum) minimum of the loss function, obtaining insufficient performance results.

As intuition, we can take a look at the figure 4.11, where on the left, the evolution of three types of batch size loss curves (arriving to the minimum) are plotted. The blue one, represents a batch of the same size of the dataset, thus making only one update per epoch, a smother curve with a much less noisy evolution. Although it seems the best approach, the detail is in the time and space it takes to update the weights, since we have a large number of samples, we have to compute a vast amount of operations. Moreover, commonly is impracticable that a complete dataset fits into a modern GPU memory.

The purple curve, is for the case where we realize one-sample updates, something that reflects, on extreme, what happened during the training of our network. Finally, the green curve shows the daily situation of most deep learning trainings, where the batch size is found in balance with the number of updates per epoch. Although there are frequent updates, they are not too much for firing divergence, at the same time a reasonable time is taken for finding the error.



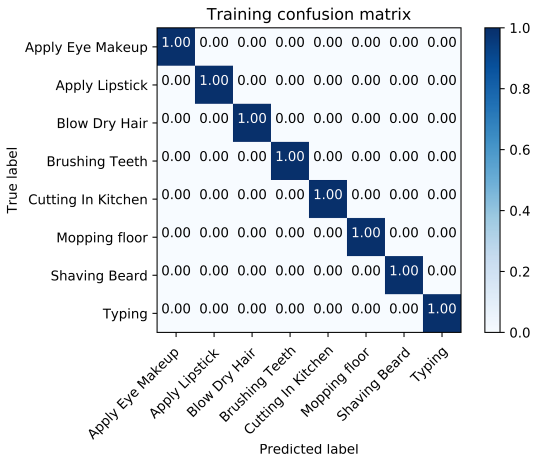
(a) Different batch size methods for finding the minimum (red).



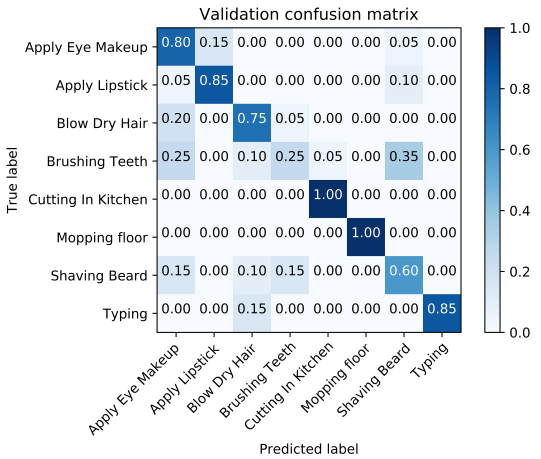
(b) Relationship between bath sizes and error for a sample dataset. Y axis is accuracy and X axis is the epoch.

Figure 4.11: Effects of batch sizes when training.

In order to better visualize where the network guesses right or wrong, we can make use of the training and validation confusion matrices 4.12, where in each cell we can see the percentage of true positives for the class in the cell row. For example, in the validation matrix, 35% percent of the times we see the class *Brushing teeth*, the network sees it as Shaving Beard. Moreover, we can note that by obtaining the trace of a confusion matrix (summation over the diagonal) and dividing by the number of classes, we retrieve the final accuracy.



(a) Training matrix.



(b) Validation matrix.

Figure 4.12: Confusion matrices for the proposed dataset.

Easly, we notice what we determined before, the training set is overfitted, since for all diagonal cells the confusion matrix reports a 100% value (normalized between 0 and 1). On the other hand, when analyzing validation matrix, we can see that the network mostly fails when the classes are very similar. For example:

- *Apply Eye Makeup* is confounded 15% of the times with *Apply Lipstick*, since both use some kind of hand stick and cover zones of the face vertically near between each other, is logic to think that they are more difficult to differentiate.
- *Apply Eye Makeup* and *Shaving Beard* follow a similar error pattern, since in both actions there is hand movement over the zone of the mouth and arm movement around the whole face.

In other cases, the contrary can happen, when the action is easily differentiable from others, this mostly happens with two actions, *Mopping the floor* which usually happens in a room, and *Cutting in kitchen* where the camera focuses on the knife and the cutting table area.

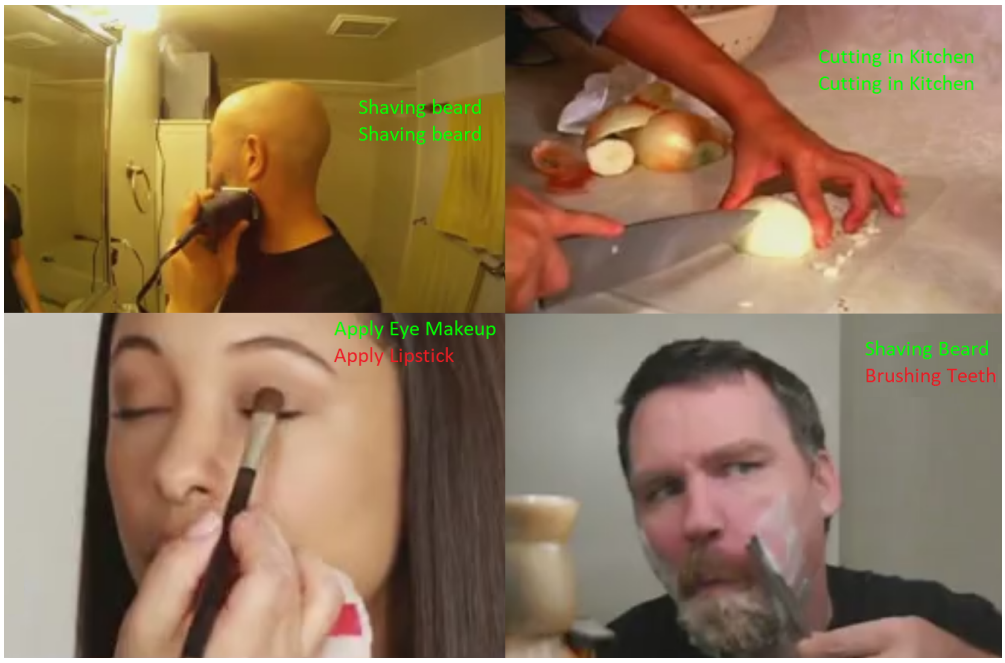


Figure 4.13: Class labels and network predictions: First line is correct label, second line is the predicted one, green if correct or red if not.

4.5 Conclusion

As we can see, it is clear that all three proposals are good candidates for selection as a way for speeding up the action recognition pipeline. On the other hand, some of them are still not mature enough for being considered in a production environment. Notwithstanding, they have shown its effectivity. For this, we should consider these technologies as a promising starting point that will allow us build and pave the way towards a more efficient and robust real-time deep learning action recognition system.

Chapter 5

Conclusion

This chapter summarizes the conclusions drawn from this work. It is divided into two sections. First, in Section 5.1 we sum up the work done in this Thesis. Then, in Section 5.3 possible improvements as well as future lines of research are explored.

5.1 Conclusions

In this Thesis, we have started with an extensive review of the different approaches that have been taken in order to solve the problem of action recognition. First, we analyzed the limitations that many traditional methods based on characteristics extractions had. Mainly, difficulties with image occlusions, lightning conditions, and multiple points of view.

On the other hand, when machine learning methods started to gain popularity due to their outstanding results in other research areas, they where applied to the task at hand, specifically through the use of SVMs (not exclusive of this research area) and more refined motion-related representations such as optical flow, which rapidly became one of the most effectives ways to describe motion. In these times, the need for large amounts of data in order to properly train the models was also noticed. Something that gained further relevance with the apparition of new deep learning techniques, which until now are the most effective methods that have been developed.

Such quality can be observed at the different works analyzed, where one particular type of neural network, CNN, stands out of the crowd. Specifically when they are applied in a two-stream fashion, in which information from the appearance (RGB) and motion (optical flow) are combined. Moreover, the apparition of recurrent networks has also allowed important developments in the field of action recognition, where their effectivity was further boosted when combined with their convolutional counterpart. Despite of this, their sequential nature usually makes them to not be impractical in real time applications.

For the most recent techniques, we have analyzed how the TSN approach has marked an inflexion point in the task at hand. Instead of obtaining multiple frames for each video, by randomly extracting three of them evenly spaced, in conjunction with their optical flow, a network can robustly learn the representation of each action category present in the dataset. Interestingly given this simplicity, the fact that with little information of an action, high quality predictions can be made, had been hinted in humans long ago. In the perception study of [55], subjects could detect without effort a range of motion patterns with a small numbers of moving light dots presented in a reminiscent human contour, as it can be observed in Figure 5.1.

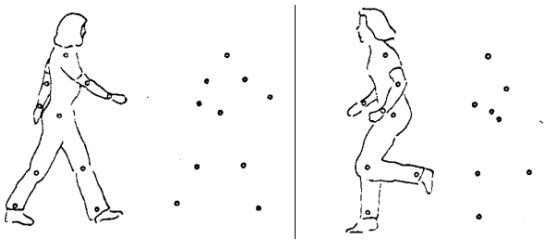


Figure 5.1: Light dots and their human contour. Walking and running actions (Reproduced from [55]).

Regarding data, we have explored how much and variated has to be for properly training deep networks. In spite of this, we have taken a look at a wide range of action recognition datasets, together with their purposes and possible applications.

Finally, we have focused our attention on different ways for accelerating the training and inference processes of a modern video-based action recognition pipeline. First, the use of a TSN framework, since it requires small amounts of data as an input. Secondly, the use of MotionNet from the HTS work, in order to achieve real-time optical flow computation times, adapt its representation for action recognition. Third, the use of the recent NVVL for reducing the cost of IO operations, save storage space, and speedup the whole pipeline by directly decoding videos on the GPU. The experiments code can be found at GitHub.[1]

5.2 Highlights

The highlights of the work are the following:

- In-depth study of the problem of action recognition over the years, from the 90th century until today.
- Review and summary of existing action human action video datasets.
- Exploration of the use of deep learning models for the task of understanding human actions.
- Search of possible improvements for accelerating the speed of a modern action recognition system.
- Proof of concept and analysis of some of the proposed acceleration techniques.

[1] https://github.com/Ivorra/tfg

5.3 Future Work

Due to the time constraints imposed on this project, many possible improvements and ideas were left out for future works. Here we summarize them to conclude this Thesis:

- Regarding NVVL, several improvements can be made: Fix the non-freeing memory issue, further explore why the loader stops working after certain videos. Use the C++ API for directly decode specific frames and call this function from PyTorch.

  Use the Cupy wrapper, since recently developers have implemented the DLPack tensor in memory intermediary representation[2], being able to convert back and forth tensor objects from this framework to any other that supports DLPack, like PyTorch (without time performance losses).

- Complete the implementation of the converted MotionNet, since actually it has different input and output shapes than the original TSN project. Furthermore, we could replicate the architecture in PyTorch and train it again.

- Explore other ways of extracting optical flow in real-time. One newly approach, seemingly not investigated yet, is implementing optical flow algorithms in the Tensor Comprehensions[3][56] framework. Basically, a set of high level abstraction tools that allow to express tensor operations in an extended Einstein notation[4]. Then, they are compiled into an intermediary code, which through evolutionary algorithms that modify the compiler parameters, different GPU code versions are generated, have their execution time measured, and select the best performing ones. Stemming from them, new generations are created, until achieving time convergence (there is no more room for improvement). They have demonstrated to be on par with manual code implementations, as well as outperforming other ones since the optimizations are done ad-hoc, for the hardware in which they are executing.

- Explore other network approaches, like Convolutional Recurrent Networks or Attention mechanisms with and without recurrence.

[2] https://github.com/NVIDIA/nvvl/issues/29

[3] https://research.fb.com/announcing-tensor-comprehensions/

[4] https://obilaniu6266h16.wordpress.com/2016/02/04/einstein-summation-in-numpy/

Bibliography

[1] Berthold K P Horn and Brian G Rhunck. 'Determining Optical Flow'. In: (1981). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.562&rep=rep1&type=pdf.

[2] John Canny. 'A Computational Approach to Edge Detection'. In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* 6 (1986). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf.

[3] L R Rabiner and B H Juang. 'An Introduction to Hidden Markov Models'. In: (1986). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.127.2942&rep=rep1&type=pdf.

[4] R. Bellman and R. Kalaba. 'On adaptive control processes'. In: *IRE Transactions on Automatic Control* 4.2 (Nov. 1959), pp. 1–9. ISSN: 0096-199X. DOI: 10.1109/TAC.1959.1104847. URL: http://ieeexplore.ieee.org/document/1104847/.

[5] C. Myers, L. Rabiner and A. Rosenberg. 'Performance tradeoffs in dynamic time warping algorithms for isolated word recognition'. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.6 (Dec. 1980), pp. 623–635. ISSN: 0096-3518. DOI: 10.1109/TASSP.1980.1163491. URL: http://ieeexplore.ieee.org/document/1163491/.

[6] Gary A. Churchill. 'Stochastic models for heterogeneous DNA sequences'. In: *Bulletin of Mathematical Biology* 51.1 (Jan. 1989), pp. 79–94. ISSN: 0092-8240. DOI: 10.1007/BF02458837. URL: http://link.springer.com/10.1007/BF02458837.

[7] D M Gavrila. 'The Visual Analysis of Human Movement: A Survey'. In: *Computer Vision and Image Understanding* 73.1 (1999), pp. 82–98. URL: http://www.idealibrary.com.

[8] J. Yamato, J. Ohya and K. Ishii. 'Recognizing human action in time-sequential images using hidden Markov model'. In: *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Comput. Soc. Press, 1992, pp. 379–385. ISBN: 0-8186-2855-3. DOI: 10.1109/CVPR.1992.223161. URL: http://ieeexplore.ieee.org/document/223161/.

[9] K. Rohr. 'Towards Model-Based Recognition of Human Movements in Image Sequences'. In: *CVGIP: Image Understanding* 59.1 (Jan. 1994), pp. 94–115. ISSN: 1049-9660. DOI: 10.1006/CIUN.1994.1006. URL: https://www.sciencedirect.com/science/article/pii/S1049966084710060.

[10] Yan Guot, Gang Xutt and Saburo Tsujitt. 'Understanding Human Motion Patterns'. In: (1994). URL: http://cyber.sci-hub.tw/MTAuMTEwOS9pY3ByLjE5OTQuNTc2OT10.1109@ICPR.1994.576929.pdf.

[11] Lee Campbell and Aaron Bobick. 'Recognition of Human Body Motion Using Phase Space Constraints'. In: (1995), pp. 624–630. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3159&rep=rep1&type=pdf.

[12] Efros, Berg, Mori et al. 'Recognizing action at a distance'. In: *Proceedings Ninth IEEE International Conference on Computer Vision*. IEEE, 2003, pp. 726–733. ISBN: 0-7695-1950-4. DOI: 10.1109/ICCV.2003.1238420. URL: http://ieeexplore.ieee.org/document/1238420/.

[13] Christian Schüldt, Ivan Laptev and Barbara Caputo. 'Recognizing Human Actions: A Local SVM Approach *'. In: (2004). URL: https://s3.amazonaws.com/academia.edu.documents/33016569/2004_icpr_schuldt.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1515145581&Signature=1FsbVu4QBQLtbEaOCi3BNFQYSBQ%3D&response-content-disposition=inline%3B%20filename%3DRecognizing_Human_Actions_A_.

[14] Konrad Schindler BIWI, Eth Zürich, Luc Van Gool et al. 'Action Snippets: How many frames does human action recognition require?' In: (2008). URL: https://www.vision.ee.ethz.ch/publications/papers/proceedings/eth_biwi_00532.pdf.

[15] Marcin Marszalek, Ivan Laptev, Cordelia Schmid et al. 'Actions in Context'. In: (2009), pp. 2929–2936. DOI: 10.1109/CVPR.2009.5206557>. URL: https://hal.inria.fr/file/index/docid/548645/filename/MarszalekLaptevSchmid-CVPR09-ActionsContext.pdf.

[16] Shuiwang Ji, Wei Xu, Ming Yang et al. 'IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 3D Convolutional Neural Networks for Human Action Recognition'. In: (2013). URL: http://www.cs.odu.edu/~sji/papers/pdf/Ji_TPAMI2012.pdf.

[17] Karen Simonyan and Andrew Zisserman. 'Two-Stream Convolutional Networks for Action Recognition in Videos'. In: (2014). URL: http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos.pdf.

[18] Guilhem Chéron and Ivan Laptev. 'P-CNN: Pose-based CNN Features for Action Recognition'. In: (2015). URL: http://openaccess.thecvf.com/content_iccv_2015/papers/Cheron_P-CNN_Pose-Based_CNN_ICCV_2015_paper.pdf.

[19] Martín Abadi, Ashish Agarwal, Paul Barham et al. 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems'. In: (2015). URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf.

[20] François Chollet and others. *Keras*. https://github.com/keras-team/keras. 2015.

[21] Amit Agarwal, Eldar Akchurin, Chris Basoglu et al. 'An Introduction to Computational Networks and the Computational Network Toolkit'. In: (2015). URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2014/08/CNTKBook-20160217.pdf.

[22] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi et al. 'Theano: A Python framework for fast computation of mathematical expressions'. In: (2016). URL: https://arxiv.org/pdf/1605.02688.pdf.

[23] Adam Paszke, Sam Gross, Soumith Chintala et al. 'Automatic differentiation in PyTorch'. In: *NIPS-W*. 2017.

[24] Ronan Collobert, Koray Kavukcuoglu and Clément Farabet. 'Torch7: A Matlab-like Environment for Machine Learning'. In: (2011). URL: https://infoscience.epfl.ch/record/192376/files/Collobert_NIPSWORKSHOP_2011.pdf.

[25] Bingbing Ni, Gang Wang and Pierre Moulin. 'RGBD-HuDaAct: A color-depth video database for human daily activity recognition'. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE, Nov. 2011, pp. 1147–1153. ISBN: 978-1-4673-0063-6. DOI: 10.1109/ICCVW.2011.6130379. URL: http://ieeexplore.ieee.org/document/6130379/.

[26] Lu Xia, Chia Chih Chen and J. K. Aggarwal. 'View invariant human action recognition using histograms of 3D joints'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2012. ISBN: 9781467316118. DOI: 10.1109/CVPRW.2012.6239233.

[27] Lu Xia, Ilaria Gori, J K Aggarwal et al. 'Robot-Centric Activity Recognition from First-Person RGB-D Videos'. In: *2015* (). URL: http://cvrc.ece.utexas.edu/lu/WACV2015_camera_ready_Lu.pdf.

[28] Ferda Ofli, Rizwan Chaudhry, Gregorij Kurillo et al. 'Berkeley MHAD: A comprehensive Multimodal Human Action Database'. In: *2013 IEEE Workshop on Applications of Computer Vision (WACV)*. IEEE, Jan. 2013, pp. 53–60. ISBN: 978-1-4673-5054-9. DOI: 10.1109/WACV.2013.6474999. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6474999.

[29] Hamed Pirsiavash and Deva Ramanan. 'Detecting Activities of Daily Living in First-person Camera Views'. In: (2012). URL: https://www.csee.umbc.edu/~hpirsiav/papers/adl_cvpr12.pdf.

[30] Amir Shahroudy, Jun Liu, Tian-Tsong Ng et al. 'NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis'. In: (2016). URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Shahroudy_NTU_RGBD_A_CVPR_2016_paper.pdf.

[31] Khurram Soomro, Amir Roshan Zamir and Mubarak Shah. 'UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild'. In: (2012). URL: http://crcv.ucf.edu/data/UCF101.php.

[32] Yuya Yoshikawa, Jiaqing Lin and Akikazu Takeuchi. 'STAIR Actions: A Video Dataset of Everyday Home Actions'. In: (2018). URL: https://arxiv.org/pdf/1804.04326.pdf.

[33] Dima Damen, Hazel Doughty, Giovanni Maria Farinella et al. 'Scaling Egocentric Vision: The EPIC-KITCHENS Dataset'. In: (2018). URL: https://arxiv.org/pdf/1804.02748.pdf.

[34] Nicholas Rhinehart and Kris M Kitani. 'First-Person Activity Forecasting with Online Inverse Reinforcement Learning'. In: (). URL: https://arxiv.org/pdf/1612.07796.pdf.

[35] Tahmida Mahmud, Mahmudul Hasan and Amit K Roy-Chowdhury. 'Joint Prediction of Activity Labels and Starting Times in Untrimmed Videos'. In: (). URL: http://openaccess.thecvf.com/content_ICCV_2017/papers/Mahmud_Joint_Prediction_of_ICCV_2017_paper.pdf.

[36] Gurkirt Singh, Suman Saha, Michael Sapienza et al. 'Online Real-time Multiple Spatiotemporal Action Localisation and Prediction'. In: (). URL: https://arxiv.org/pdf/1611.08563v6.pdf.

[37] Yu Kong, Zhiqiang Tao and Yun Fu. 'Deep Sequential Context Networks for Action Prediction'. In: (). DOI: 10.1109/CVPR.2017.390. URL: https://www.researchgate.net/profile/Yu_Kong6/publication/318659269_Deep_Sequential_Context_Networks_for_Action_Prediction/links/599e11a1aca272dff12fddc9/Deep-Sequential-Context-Networks-for-Action-Prediction.pdf.

[38] Kuo-Hao Zeng, William B Shen, De-An Huang et al. 'Visual Forecasting by Imitating Dynamics in Natural Sequences'. In: (). URL: https://arxiv.org/pdf/1708.05827.pdf.

[39] Jian-Fang Hu, Wei-Shi Zheng, Lianyang Ma et al. 'Real-Time RGB-D Activity Prediction by Soft Regression'. In: (). DOI: 10.1007/978-3-319-46448-0. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-319-46448-0_17.pdf.

[40] Colin Lea, Michael D Flynn, René Vidal et al. 'Temporal Convolutional Networks for Action Segmentation and Detection'. In: (). URL: http://openaccess.thecvf.com/content_cvpr_2017/papers/Lea_Temporal_Convolutional_Networks_CVPR_2017_paper.pdf.

[41] Shyamal Buch, Victor Escorcia, Chuanqi Shen et al. 'SST: Single-Stream Temporal Action Proposals'. In: (). URL: http://openaccess.thecvf.com/content_cvpr_2017/papers/Buch_SST_Single-Stream_Temporal_CVPR_2017_paper.pdf.

[42] João Carreira and Andrew Zisserman. 'Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset'. In: (). URL: http://openaccess.thecvf.com/content_cvpr_2017/papers/Carreira_Quo_Vadis_Action_CVPR_2017_paper.pdf.

[43] Christoph Feichtenhofer, Axel Pinz and Richard P Wildes. 'Spatiotemporal Multiplier Networks for Video Action Recognition'. In: (). URL: http://openaccess.thecvf.com/content_cvpr_2017/papers/Feichtenhofer_Spatiotemporal_Multiplier_Networks_CVPR_2017_paper.pdf.

[44] Achal Dave, Olga Russakovsky and Deva Ramanan. 'Predictive-Corrective Networks for Action Detection'. In: (). URL: http://openaccess.thecvf.com/content_cvpr_2017/papers/Dave_Predictive-Corrective_Networks_for_CVPR_2017_paper.pdf.

[45] Gunnar A Sigurdsson, Santosh Divvala, Ali Farhadi et al. 'Asynchronous Temporal Fields for Action Recognition'. In: (). URL: http://openaccess.thecvf.com/content_cvpr_2017/papers/Sigurdsson_Asynchronous_Temporal_Fields_CVPR_2017_paper.pdf.

[46] Limin Wang, Yuanjun Xiong, Zhe Wang et al. 'Temporal Segment Networks for Action Recognition in Videos'. In: (2017). URL: https://arxiv.org/pdf/1705.02953.pdf.

[47] Mohammad Sadegh Aliakbarian, Fatemeh Sadat Saleh, Mathieu Salzmann et al. 'Encouraging LSTMs to Anticipate Actions Very Early'. In: (2017). URL: https://arxiv.org/pdf/1703.07023.pdf.

[48] Bolei Zhou, Aditya Khosla, Agata Lapedriza et al. 'Learning Deep Features for Discriminative Localization'. In: (2016). URL: http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf.

[49] Yi Zhu, Zhenzhong Lan, Shawn Newsam et al. 'Hidden Two-Stream Convolutional Networks for Action Recognition'. In: (2017). URL: https://arxiv.org/pdf/1704.00389.pdf.

[50] Joe Yue, Hei Ng, Jonghyun Choi et al. 'ActionFlowNet: Learning Motion Representation for Action Recognition'. In: (2016). URL: https://arxiv.org/pdf/1612.03052.pdf.

[51] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia et al. 'FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks'. In: (2017). URL: https://arxiv.org/pdf/1612.01925.pdf.

[52] Brendan Duke. *Lintel: Python Video Decoding*. https://github.com/dukebw/lintel. 2018.

[53] Jared Casper, Jon Barker and Bryan Catanzaro. *NVVL: NVIDIA Video Loader*. https://github.com/NVIDIA/nvvl. 2018.

[54] Jared Casper and Jon Barker. 'NVVL Accelerates Machine Learning on Video Datasets'. In: (May 2018). URL: https://devblogs.nvidia.com/acclerate-machine-learnng-nvvl/.

[55] Gunnar Johansson. 'Visual perception of biological motion and a model for its analysis"' in: *Perception & Psychophysics* 14.2 (1973), pp. 201–211. URL: https://link.springer.com/content/pdf/10.3758/BF03212378.pdf.

[56] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis et al. 'Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions'. In: (2018). URL: https://arxiv.org/pdf/1802.04730.pdf.